21世纪高等学校规划教材 | 软件工程





实用软件测试

李炳森 主编

清华大学出版社

21世纪高等学校规划教材·软件工程

实用软件测试

李炳森　主编

清华大学出版社
北　京

内 容 简 介

本书突出实用特色，讲述软件测试的相关概念、方法和技能，全书分为基础篇、技术篇和工具篇三大部分：基础篇讲述软件测试与软件质量的基础理论，为后面的学习奠定一定的理论基础；技术篇讲述了面向传统开发过程、面向软件工程层面的软件测试和自动化测试以及敏捷测试；工具篇介绍黑盒测试工具与白盒测试工具、性能测试工具与安全测试工具以及测试管理工具，并选取常用软件测试工具讲述其使用方法。

本书重视实践能力和操作能力的培养，内容翔实、循序渐进、图文并茂、实用性强，并在案例讲述过程中穿插相关的基础知识和基本理论介绍，做到理论与实践相结合，方法与应用相结合，读者可在较短的时间内理解和掌握软件测试的基本概念和操作实务。

本书是软件测试相关课程的配套教材，适用于企事业等单位从事软件测试工作的人员参考学习，也可作为大中专院校计算机、软件工程、测试等相关专业师生自学、教学参考书以及社会各类培训班的即学即用教材，也适用于计算机技术与软件专业技术资格(水平)考试的继续教育。

图书在版编目(CIP)数据

实用软件测试/李炳森主编. —北京：清华大学出版社，2016
21世纪高等学校规划教材·软件工程
ISBN 978-7-302-42919-7

Ⅰ. ①实…　Ⅱ. ①李…　Ⅲ. ①软件－测试－高等学校－教材　Ⅳ. ①TP311.5

中国版本图书馆CIP数据核字(2016)第030890号

责任编辑：付弘宇　薛　阳
封面设计：
责任校对：梁　毅
责任印制：

出版发行：清华大学出版社
网　　址：http://www.tup.com.cn，http://www.wqbook.com
地　　址：北京清华大学学研大厦A座　**邮　　编**：100084
社 总 机：010-62770175　**邮　　购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载：http://www.tup.com.cn，010-62795954
印 刷 者：
装 订 者：
经　　销：全国新华书店
开　　本：185mm×260mm　**印　　张**：22.75　**字　　数**：564千字
版　　次：2016年4月第1版　**印　　次**：2016年4月第1次印刷
印　　数：1～　000
定　　价：　.00元

产品编号：056517-01

本书编委会

出版说明

随着我国改革开放的进一步深化，高等教育也得到了快速发展，各地高校紧密结合地方经济建设发展需要，科学运用市场调节机制，加大了使用信息科学等现代科学技术提升、改造传统学科专业的投入力度，通过教育改革合理调整和配置了教育资源，优化了传统学科专业，积极为地方经济建设输送人才，为我国经济社会的快速、健康和可持续发展以及高等教育自身的改革发展做出了巨大贡献。但是，高等教育质量还需要进一步提高以适应经济社会发展的需要，不少高校的专业设置和结构不尽合理，教师队伍整体素质亟待提高，人才培养模式、教学内容和方法需要进一步转变，学生的实践能力和创新精神亟待加强。

教育部一直十分重视高等教育质量工作。2007 年 1 月，教育部下发了《关于实施高等学校本科教学质量与教学改革工程的意见》，计划实施"高等学校本科教学质量与教学改革工程(简称'质量工程')"，通过专业结构调整、课程教材建设、实践教学改革、教学团队建设等多项内容，进一步深化高等学校教学改革，提高人才培养的能力和水平，更好地满足经济社会发展对高素质人才的需要。在贯彻和落实教育部"质量工程"的过程中，各地高校发挥师资力量强、办学经验丰富、教学资源充裕等优势，对其特色专业及特色课程(群)加以规划、整理和总结，更新教学内容、改革课程体系，建设了一大批内容新、体系新、方法新、手段新的特色课程。在此基础上，经教育部相关教学指导委员会专家的指导和建议，清华大学出版社在多个领域精选各高校的特色课程，分别规划出版系列教材，以配合"质量工程"的实施，满足各高校教学质量和教学改革的需要。

为了深入贯彻落实教育部《关于加强高等学校本科教学工作，提高教学质量的若干意见》精神，紧密配合教育部已经启动的"高等学校教学质量与教学改革工程精品课程建设工作"，在有关专家、教授的倡议和有关部门的大力支持下，我们组织并成立了"清华大学出版社教材编审委员会"(以下简称"编委会")，旨在配合教育部制定精品课程教材的出版规划，讨论并实施精品课程教材的编写与出版工作。"编委会"成员皆来自全国各类高等学校教学与科研第一线的骨干教师，其中许多教师为各校相关院、系主管教学的院长或系主任。

按照教育部的要求，"编委会"一致认为，精品课程的建设工作从开始就要坚持高标准、严要求，处于一个比较高的起点上；精品课程教材应该能够反映各高校教学改革与课程建设的需要，要有特色风格、有创新性(新体系、新内容、新手段、新思路，教材的内容体系有较高的科学创新、技术创新和理念创新的含量)、先进性(对原有的学科体系有实质性的改革和发展，顺应并符合 21 世纪教学发展的规律，代表并引领课程发展的趋势和方向)、示范性(教材所体现的课程体系具有较广泛的辐射性和示范性)和一定的前瞻性。教材由个人申报或各校推荐(通过所在高校的"编委会"成员推荐)，经"编委会"认真评审，最后由清华大学出版

社审定出版。

目前，针对计算机类和电子信息类相关专业成立了两个“编委会”，即“清华大学出版社计算机教材编审委员会”和“清华大学出版社电子信息教材编审委员会”。推出的特色精品教材包括：

（1）21世纪高等学校规划教材·计算机应用——高等学校各类专业，特别是非计算机专业的计算机应用类教材。

（2）21世纪高等学校规划教材·计算机科学与技术——高等学校计算机相关专业的教材。

（3）21世纪高等学校规划教材·电子信息——高等学校电子信息相关专业的教材。

（4）21世纪高等学校规划教材·软件工程——高等学校软件工程相关专业的教材。

（5）21世纪高等学校规划教材·信息管理与信息系统。

（6）21世纪高等学校规划教材·财经管理与应用。

（7）21世纪高等学校规划教材·电子商务。

（8）21世纪高等学校规划教材·物联网。

清华大学出版社经过三十多年的努力，在教材尤其是计算机和电子信息类专业教材出版方面树立了权威品牌，为我国的高等教育事业做出了重要贡献。清华版教材形成了技术准确、内容严谨的独特风格，这种风格将延续并反映在特色精品教材的建设中。

清华大学出版社教材编审委员会
联系人：魏江江
E-mail：weijj@tup.tsinghua.edu.cn



随着软件应用领域的不断深入、设计复杂程度的逐步扩大、开发周期的不断缩短、质量要求的不断提高，软件企业也面临着巨大的挑战。因此，加强软件测试过程和技术，可以有效地保证软件质量。这种观念正在被更多的软件企业人士所理解、接受和实施，也是软件企业快速发展的必经之路。

软件测试是软件工程中重要的组成部分，是软件质量保证的关键步骤，是保证软件质量的必要依据，对保证软件质量具有重要意义。关于软件测试的研究结果表明：软件中存在的问题发现得越早，软件的开发费用就越低；在编码后修改软件缺陷的成本是编码前的10倍，在产品交付后修改软件缺陷的成本是交付前的10倍；软件质量越高，软件发布后的维护费用就越低。软件测试费用占整个软件工程所有研发费用的50%以上。

掌握好的软件测试方法是提升软件企业工作质量的基础；编制好的软件测试过程是提升软件企业工作质量的前提；软件测试是提升软件企业硬件水平的根本手段，改进软件组织管理过程的必要途径。为了培养具备软件工程思想和技术以及相应开发经验的计算机和软件人才，国家近年来一直十分重视软件工程相关课程的建设工作。

在此背景下，《实用软件测试》一书应运而生。该书突出实用特色，讲述软件测试的相关概念、方法和技能。

全书分为三大部分：基础篇、技术篇、工具篇。基础篇讲述软件测试与软件质量的基础理论，为后面的学习奠定了一定的理论基础；技术篇讲述面向传统开发过程、面向软件工程层面的软件测试和自动化测试以及敏捷测试；工具篇介绍黑盒测试工具与白盒测试工具、性能测试工具与安全测试工具以及测试管理工具，并选取了常用软件测试工具讲述其使用方法。本书中提出了一个可操作性强、易于上手的解决办法，能够帮助读者清晰地了解软件测试的整个过程，理解如何做好软件测试工作。

本书围绕着实际的软件项目展开，理论联系实际，给出了具有很强实践性的软件测试具体建议。在学术上，本书的主要线索是编者在软件测试基本知识的基础上，结合长期从事软件测试工作的实际经验，剪裁出来的一个针对软件测试的实用方法。

本书重视实践能力和操作能力的培养，内容翔实、条例清晰、循序渐进、图文并茂、理论扎实、实用性强，并在案例讲述过程中穿插相关的基础知识和基本理论介绍，做到理论与实践相结合，方法与应用相结合，读者可以最快的速度理解和掌握软件测试的基本概念和操作实务，可以帮助软件测试人员快速应用到工作中，有效提高项目质量和效率。

本书是软件测试相关课程的配套教材，适用于企事业等单位从事软件测试工作的人员参考学习，也可作为大中专院校计算机、软件工程、测试等相关专业师生自学、教学参考书以及社会各类培训班的即学即用教材，也适用于计算机技术与软件专业技术资格(水平)考试的继续教育。

本书具有以下特点。

(1) 编者多年从事软件测试的经验总结,来源于实践,更具实用性。

(2) 理论联系实际,围绕真实的软件项目展开。

(3) 读者只要完整地研读了本书,就切实掌握了整个软件测试的全貌。

(4) 提供完整的软件测试操作实务,读者可直接用于实践。

(5) 循序渐进。本书将内容分为基础篇、技术篇、工具篇三大部分,层次分明,便于循序渐进地讲述知识,便于读者学习与理解。

(6) 实用性强。本书所选案例贯穿全书,以案例驱动;又对不同的测试方法和技术选用不同的案例,做到有所针对;同时介绍了工具使用和文档撰写,具有很强的实用性。

(7) 理论结合。本书在案例介绍、工具介绍过程中穿插相关的理论知识和基本方法,使基础知识更具体形象,同时也更容易被理解和应用。

(8) 实时性强。本书所选案例均是近年来的真实案例,可以代表当代技术特征和需求环境;本书介绍的工具均是当前常见的软件测试工具;面向对象测试策略的内容符合软件测试技术的发展方向。

1. 章节内容与编写情况介绍

软件质量是软件产品的生命线;软件测试是保证软件质量的必要手段。本书力求通过循序渐进、图文并茂的方式使读者能以最快的速度理解和掌握基本概念和应用方法。全书共计10章,各章的内容安排如下。

第1章介绍软件缺陷与软件测试的基本概念、软件测试技术的发展历史与现状、软件测试过程、软件测试与软件开发的联系及其重要性与实质。

第2章介绍质量与软件质量的基本概念、软件质量模型、标准的发展情况以及软件质量、软件质量保证与软件测试的区别与联系。

第3章介绍面向传统开发过程的软件测试,涉及的内容有软件测试模型、软件生命周期、单元测试、集成测试、系统测试。

第4章介绍面向软件工程层面的软件测试,涉及的内容有面向对象的测试、面向方面的测试和面向SOA的测试。

第5章介绍自动化测试的有关内容,如自动化测试的优点、基本原则、实现策略与步骤等。

第6章结合项目实例介绍敏捷测试每个阶段的主要测试活动,分析每个主要测试活动的前提条件和目标任务,推荐最佳的解决方案。

第7章介绍软件测试工具的总体情况,涉及的内容有工具角度分类、常见测试工具的对比、测试工具的选择方法。

第8章介绍面向功能的测试工具。

第9章介绍面向质量属性角度的测试工具,包括性能测试工具和安全测试工具两部分内容。

第10章介绍测试管理工具,包括缺陷管理工具和综合管理工具两部分内容。

附录 介绍软件测试的相关术语和常见问题,以及全国计算机技术与软件专业技术资格(水平)考试软件评测师的最新考试大纲和模拟试题、参考答案、评分标准。

本书的末尾还附有模拟试题与解答以供读者熟悉和巩固所学知识,作者将长期实践的

经验融入其中,相信必会使读者受益匪浅。在本书中,强调知识重点并给予读者练习的机会,最好能够详细阅读并亲身实践。

本书由李炳森任主编,并率领编委会的委员们对全书进行了统筹、规划、审校、修改和协调,委员们对全书的编写也提出了许多宝贵的意见或建议,并为本书提供了很多有价值的素材。其中,第 1、2 章由李炳森、田杰、段成峰、田国栋、刘红云共同编写,第 3~5 章由班日哲、张华、李炳森共同编写,第 6 章由陈晓颖、李炳森共同编写,第 7~10 章由班日哲、张华、李炳森共同编写,附录 A、附录 B 由梁金静负责收集整理,附录 C 由李炳森负责收集整理,附录 D、附录 E 由张华、李炳森共同编写。

2. 技术支持

本书由彦哲研究院组织编写。彦哲研究院是北京彦哲信息技术服务有限公司的专家顾问委员会,组建于 2006 年 5 月 18 日,其前身为管理与信息化联盟,由多位资深的信息化管理专家自愿发起,是信息化管理先进理论与最佳实践有机结合的推进者,是跨地区、跨行业的专业服务组织。其宗旨是加强全国信息化管理领域的交流与合作,积极推进国家信息化建设和管理工作的发展。彦哲研究院总部设在北京,现有研究员五百余名,分布在全国五十多个大中城市的多个行业领域。

有关本书的意见反馈和咨询,读者可登录由彦哲研究院主办的信息化管理专家网(http://www.yima.org.cn)与编者进行交流。

与本书配套的 PPT 课件等教学资源可以从清华大学出版社网站 www.tup.com.cn 下载,关于本书与课件使用中的任何问题,请联系 fuhy@tup.tsinghua.edu.cn。

3. 致谢

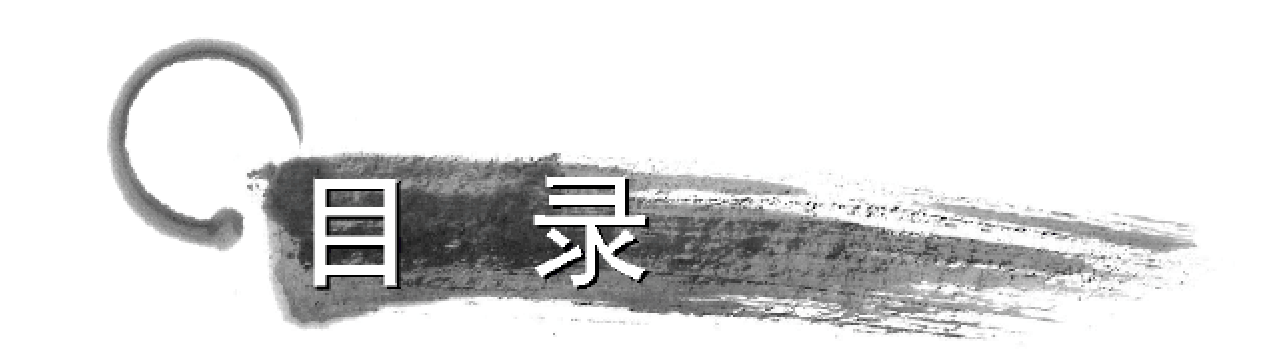
在本书的编写过程中,参考了许多相关的资料和书籍,在此恕不一一列举(详见参考文献),编者在此对这些参考文献的作者表示诚挚的感谢。

在本书的出版过程中,来自清华大学、北京大学、天津大学、北京交通大学、武汉纺织大学、中国国际贸易学会服务外包实务教学工作委员会、全国服务外包岗位专业考试中心、中国外包世界(香港)有限公司、北京彦哲信息技术服务有限公司等单位的领导和老师们提出了许多宝贵的意见或建议,得到了商务部、教育部、工业和信息化部和中国国际贸易学会领导们的悉心指导,也得到了清华大学出版社给予的支持和帮助,在此向所有关心和支持本书出版的人士表示感谢。

由于编者水平所限,书中疏漏之处在所难免,恳请各位专家和读者朋友们不吝赐教、批评指正,对此编者将深为感激。

编 者

2015 年 10 月



基 础 篇

技 术 篇

基　础　篇



本篇内容：

- 软件测试概述
- 软件质量

第1章 软件测试概述

质量是企业的生命线。当然，软件质量就是软件企业的生命线，或者说软件质量就是软件的生命。为了保证软件的质量，人们在长期的开发过程中积累了许多经验并形成了许多行之有效的方法。但是借助这些方法，只能尽可能地减少软件中的错误和不足，却不能完全避免所有的错误。

如果把所开发出来的软件看作一个企业生产的产品，那么软件测试就相当于该企业的质量检测部分。简单地说，在编写完一段代码之后，检查其是否如所预期的那样运行，这个活动就可以看作是一种软件测试工作。新的测试理论、测试方法、测试技术手段在不断涌出，软件测试机构和组织也在迅速产生和发展，由此软件测试技术职业也同步完善和健全起来。

本章学习重点

- 了解什么是软件缺陷。
- 了解软件测试技术的发展历史与现状。
- 掌握软件测试的概念。
- 熟悉软件测试的过程。
- 熟悉软件测试与软件开发之间的关系。
- 了解软件测试的重要性和实质。

本章学习难点

了解软件测试的实质。

1.1 软件缺陷

1. 软件缺陷的起源

软件缺陷(Defect)，常常被称为 Bug。下面来了解一下软件 Bug 名称的起源。

1946 年，Grace Hopper 在发生故障的 Mark Ⅱ 计算机的继电器触点里，找到了一只被夹扁的小飞蛾，正是这只小虫子“卡”住了机器的运行。Hopper 顺手将飞蛾夹在工作笔记里，

并诙谐地把程序故障称为“Bug”。Bug 的意思是“臭虫”，而这一奇怪的称呼，后来演变成计算机行业的专业术语。虽然现代计算机再也不可能夹扁任何飞蛾，大家还是习惯地把排除程序故障叫作 Debug。

Grace Murray Hopper 被称为计算机之母，是计算机专业的先驱人物，开发了 COBOL——一种被广泛应用于商业的程序语言。

她是推动计算机普及化的功臣，被同僚称为是个“数学家、计算机科学家、社会学家、企业政治家、行销专家、计算机系统及程序设计家……还有，永远都是个预知者”。她于 1992 年逝世。

1928 年，Grace Hopper 女士从 Vassa 数学系毕业后，她拜在数学家 Oystein Ore 门下，1934 年拿到了耶鲁大学的数学硕士资格，她与计算机真正结缘应该是在她加入美国海军的 WAVES 组织后，以中尉的身份加入了在哈佛大学的研究，也是第三位加入这项研究的科学家。

1949 年，她加入了 Eckert-Mauchly 公司，一直到 1971 年正式离职。在 Eckerd-Mauchly 公司，Hopper 为第一代大规模数字计算机发明了程序语言，并一同发明了相关的编译系统——A-O。1952 年，发布了关于编译系统的第一份论文。A-O 的下一代产品被命名为 FLOW-MATIC，直接导致了 COBOL 程序语言的产生。在 COBOL 产生之前，大家一直都在用汇编语言编写程序。Hooper 成功地使程序语言的语法同自然语言的语法相类似，这样，非技术人员也可以编写代码，开启了商业程序代码的时代，将编程工作的范围从科学工程领域逐渐扩大开来。

Hopper 在海军一直供职到 1986 年，逝世前曾作为 DEC 的高级顾问工作了很短的一段时间。Hopper 是早期提出共享代码库的人士之一，并发明了用于编译软件的编译系统以及编译标准。Hopper 为 Mark Ⅰ以及后续机器 Mark Ⅱ、Mark Ⅲ编写出大量软件。

她在软件设计领域的一大“发明”，就是创造出著名的计算机术语 Bug。

2. 软件缺陷的定义

软件缺陷(Bug)，即计算机系统或者程序中存在的任何一种破坏正常运行能力的问题、错误，或者隐藏的功能缺陷、瑕疵。缺陷会导致软件产品在某种程度上不能满足用户的需要。

Error 属于缺陷的一种——内部缺陷，往往是软件本身的问题，如程序的算法错误、语法错误、数据计算不正确、数据溢出等。软件错误往往导致系统某项功能失效，或成为系统使用的故障。软件的故障、失效是指软件所提供给用户的功能或服务，不能达到用户的要求或没有达到事先设计的指标，在功能使用时中断，最后的结果或得到的结果是不正确的。软件缺陷的产生主要是由软件产品的特点和开发过程决定的。

国际标准 IEEE 729—1983 对缺陷有一个标准的定义：从产品内部看，缺陷是软件产品开发或维护过程中存在的错误、毛病等各种问题；从产品外部看，缺陷是系统所需要实现的某种功能的失效或违背。

3. 软件缺陷的类型

软件缺陷表现的形式有多种，不仅体现在功能的失效方面，还体现在其他方面。软件缺

陷的主要类型有以下几种。

(1) 软件未实现产品说明书要求的功能。功能、特性没有实现或部分实现。如设计不合理,存在缺陷。

(2) 软件出现了产品说明书指明不会出现的错误。实际结果和预期结果不一致,如运行出错,包括运行中断、系统崩溃、界面混乱,再如数据结果不正确、精度不够。

(3) 软件超出实现了产品说明书提到的功能。软件实现了产品规格说明没有提到的功能模块。

(4) 软件没有实现产品说明书虽未明确指出但应该实现的目标。

(5) 软件难以理解,不易使用,运行缓慢或者最终用户认为不好。用户不能接受的其他问题,如存取时间过长、界面不美观。

4. 软件缺陷的级别

作为软件测试员,可能所发现的大多数问题不是那么明显、严重,而是难以觉察的简单而细微的错误,有些是真正的错误,也有些不是。一般来说,问题越严重的,其优先级越高,越要得到及时的纠正。软件公司对缺陷严重性级别的定义不尽相同,但一般可以概括为以下 4 种级别。

(1) 微小的(Minor)。一些小问题如有个别错别字、文字排版不整齐等,对功能几乎没有影响,软件产品及属性仍可使用。

(2) 一般的(Major)。不太严重的错误,这样的软件缺陷虽然不影响系统的基本使用,但没有很好地实现功能,没有达到预期效果,如次要功能模块丧失、提示信息不够准确、用户界面差和操作时间长等。

(3) 严重的(Critical)。严重错误,指功能或特性没有实现,主要功能部分丧失,次要功能全部丧失,或致命的错误声明。

(4) 致命的(Fatal)。致命的错误,造成系统崩溃、死机、系统悬挂,或造成数据丢失、主要功能完全丧失等。

5. 软件缺陷的状态

软件缺陷除了严重性之外,还存在反映软件缺陷处于一种什么样的状态,便于跟踪和管理某个产品的缺陷,可以定义不同的 Bug 状态。

(1) 激活或打开:问题没有解决,存在源代码中,确认"提交的缺陷",等待处理,如测试人员新报的缺陷或者验证后 Bug 依然存在。

(2) 已修正或修复:已被开发人员检查、修复过的缺陷,通过单元测试,认为已经解决但还没有被测试人员验证。

(3) 关闭或非激活:测试人员验证已经修正的 Bug 后,确认缺陷不存在之后的状态。

(4) 重新打开:测试人员验证后,缺陷没有通过所标志的状态;或已经修改正确的问题,又重新出现错误。

(5) 推迟:可以在下一个版本中修复。

(6) 保留:由于技术原因或者第三方软件的缺陷,开发人员不能修复的缺陷。

(7) 不能重现:开发不能再现,需要测试人员检查缺陷再现的步骤。

(8) 需要更多信息：开发能再现这个缺陷，但开发人员需要一些信息，如日志文件、图片。

6. 软件缺陷的产生原因

在软件开发的过程中，软件缺陷的产生是不可避免的。那么造成软件缺陷的主要原因有哪些？可以从软件本身、团队工作和技术问题等多个方面分析，就可以了解造成软件缺陷的主要因素，比较容易确定造成软件缺陷的原因，软件缺陷的产生主要是由软件产品的特点和开发过程决定的。现归纳如下。

1) 软件本身造成的问题

(1) 需求不清晰，导致设计目标偏离客户的需求，从而引起功能或产品特征上的缺陷。

(2) 系统结构非常复杂，而又无法设计成一个很好的层次结构或组件结构，结果导致意想不到的问题或系统维护、扩充上的困难；即使设计成良好的面向对象的系统，由于对象、类太多，很难完成对各种对象、类相互作用的组合测试，而隐藏着一些参数传递、方法调用、对象状态变化等方面的问题。

(3) 对程序逻辑路径或数据范围的边界考虑不够周全，漏掉某些边界条件，造成容量或边界错误，或者引起强度或负载问题。

(4) 对一些实时应用，要进行精心设计和技术处理，保证精确的时间同步，否则容易引起时间上不协调，不一致性带来的问题。

(5) 没有考虑系统崩溃后的自我恢复或数据的异地备份、灾难性恢复等问题，从而存在系统安全性、可靠性的隐患。

(6) 系统运行环境的复杂，不仅用户使用的计算机环境千变万化，包括用户的各种操作方式或各种不同的输入数据，容易引起一些特定用户环境下的问题；在系统实际应用中，数据量很大，从而会引起强度或负载问题。

(7) 由于通信端口多、存取和加密手段的矛盾性等，会造成系统的安全性或适用性等问题。

(8) 新技术的采用，可能涉及技术或系统兼容的问题，事先没有考虑到。

(9) 文档错误、内容不正确或拼写错误。

(10) 硬件或系统软件上存在的错误。

(11) 软件开发标准或过程上的错误。

2) 团队工作造成的问题

(1) 系统需求分析时对客户的需求理解不清楚，或者和用户的沟通存在一些困难。

(2) 不同阶段的开发人员相互理解不一致。例如，软件设计人员对需求分析的理解有偏差，编程人员对系统设计规格说明书某些内容重视不够，或存在误解。

(3) 对于设计或编程上的一些假定或依赖性，相关人员没有充分沟通。

(4) 项目组成员技术水平参差不齐，新员工较多，或培训不够等原因也容易引起问题。

3) 技术问题

(1) 算法错误：在给定条件下没能给出正确或准确的结果。

(2) 语法错误：对于编译性语言程序，编译器可以发现这类问题；但对于解释性语言程

序，只能在测试运行时发现。

（3）计算和精度问题：计算的结果没有满足所需要的精度。

（4）系统结构不合理、算法选择不科学，造成系统性能低下。

（5）接口参数传递不匹配，导致模块集成出现问题。

4）项目管理的问题

（1）缺乏质量文化，不重视质量计划，对质量、资源、任务、成本等的平衡性把握不好，容易挤掉需求分析、评审、测试等时间，遗留的缺陷会比较多。

（2）系统分析时对客户的需求不是十分清楚，或者和用户的沟通存在一些困难。

（3）开发周期短，需求分析、设计、编程、测试等各项工作不能完全按照定义好的流程来进行，工作不够充分，结果也就不完整、不准确，错误较多；周期短，还给各类开发人员造成太大的压力，引起一些人为的错误。

（4）开发流程不够完善，存在太多的随机性和缺乏严谨的内审或评审机制，容易产生问题。

（5）文档不完善，风险估计不足等。

7．软件缺陷的组成与分布

软件缺陷是由很多原因造成的，如果把它们按需求分析结果——规格说明书、系统设计结果、编程的代码等归类起来，通过对缺陷分布情况的仔细分析，可以帮助我们将测试的主要精力更好地集中到最有价值的地方，图 1-1 显示了缺陷的分布情况。比较后发现，需求和架构设计规格说明书是软件缺陷出现最多的地方。可以看出，缺陷的起源及其比例如下。

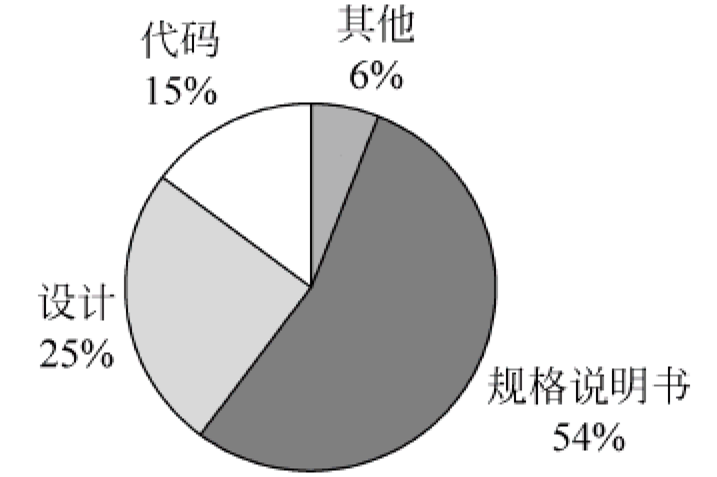


图 1-1　软件缺陷构成示意图

（1）需求和架构设计 54％。

（2）设计阶段 25％。

（3）编码阶段 15％。

（4）其他 6％。

在开发早期，错误通常是很多的，而且令人讨厌的是它们还会转移到后期。这和制造业的装配线类似，如果一个坏零件或次品被允许上线，从这点开始，包含它的组件就是“坏”的，如果该组件下了线，并出了厂门，情况就会更糟糕，必须为这个坏零件付出代价。换句话说，错误不是自封闭的，当它们转移到后面的组件中时，往往会以新的形式出现。

所有的错误都是要付出代价的。没有被发现的错误，以及那些在开发过程中很晚才发现的错误都是成本最高的，没有被发现的错误会在系统中迁移、扩散，最终导致系统失效，造成严重的财产损失，有时还会带来法律上的麻烦，系统将终身为此付出高昂的代价。

这意味着测试是贯穿开发全过程的工作，也意味着对最终产品的测试不仅是软件质量大战中的一个战役，而且不是代价最高的战役。今天，40％～70％的软件开发时间和资源都花在查错和纠错上。不幸的是大多数组织还没有一套办法来准确地计算成本。不管怎样，在使用测试资源方面任何有意义的改进都能极大地降低开发成本。

1.2 软件测试技术的发展历史与现状

1. 软件测试技术的发展历史

随着计算机的诞生——在软件行业发展初期就已经开始实施软件测试,软件测试是伴随着软件的产生而产生的,有了软件生产和运行就必然有软件测试。但早期的软件开发过程中,这一阶段还没有系统意义上的软件测试,测试的含义比较狭窄,更多的是一种类似调试的测试,将测试等同于"调试",目的是纠正软件中已经知道的故障,常常由开发人员自己完成这部分的工作。测试是没有计划和方法的,测试用例的设计和选取也都是根据测试人员的经验随机进行的,大多数测试的目的是为了证明系统可以正常运行。对测试的投入极少,测试介入得也晚,常常是等到形成代码,产品已经基本完成时才进行测试。

直到 1957 年,软件测试才开始与调试区别开来,成为一种发现软件缺陷的活动。由于一直存在着为了使我们看到产品在工作,就得将测试工作往后推一点的思想,测试仍然是后于开发的活动。在潜意识里,我们的目的是使自己确信产品能工作。

20 世纪 50 年代后期到 20 世纪 60 年代,各种高级语言相继诞生,测试的重点也逐步转入到使用高级语言编写的软件系统中,但程序的复杂性远远超过了以前。尽管如此,由于受到硬件的制约,在计算机系统中,软件仍然处于次要位置。软件正确性的把握仍然主要依赖于编程人员的技术水平。因此,这一时期软件测试的理论和方法发展比较缓慢。

到了 20 世纪 70 年代,尽管对"软件工程"的真正含义还缺乏共识,但这一词条已经频繁出现。1972 年,在北卡罗来纳大学举行了首届软件测试正式会议。1975 年,John Good Enough 和 Susan Gerhart 在 IEEE 上发表了"测试数据选择的原理(*Toward a Theory of Test Data Selection*)"的文章,软件测试才被确定为一种研究方向。而 1979 年,Glen ford Myers 的《软件测试艺术》(*The Art of Software Testing*)可算是软件测试领域的第一本最重要的专著,Myers 作为当时最好的软件测试,其定义是:"测试是为发现错误而执行的一个程序或者系统的过程"。Myers 以及他的同事们在 20 世纪 70 年代的工作是测试过程发展的里程碑。

20 世纪 70 年代以后,随着计算机处理速度的提高,存储器容量的快速增加,软件在整个计算机系统中的地位变得越来越重要。随着软件开发技术的成熟和完善,软件的规模也越来越大,复杂度也大为增加。因此,软件的可靠性面临着前所未有的危机,给软件测试工作带来了更大的挑战,很多测试理论和测试方法应运而生,逐渐形成了一套完整的体系,培养和造就了一批批出色的测试人才。

直到 20 世纪 80 年代早期,"质量"的号角才开始吹响。软件测试定义发生了改变,测试不单纯是一个发现错误的过程,而且包含软件质量评价的内容。软件开发人员和测试人员开始坐在一起探讨软件工程和测试问题。制定了各类标准,包括 IEEE(Institute of Electrical and Electronic Engineers)标准、美国 ANSI(American National Standard Institute)标准以及 ISO(International Standard Organization)国际标准。1983 年,Bill Hetzel 在《软件测试完全指南》(*Complete Guide of Software Testing*)一书中指出:"测试是以评价一个程序或者系统属性为目标的任何一种活动,测试是对软件质量的度量"。

Myers 和 Hetzel 的定义至今仍被引用。

20 世纪 90 年代，测试工具终于盛行起来。人们普遍意识到工具不仅是有用的，而且要对今天的软件系统进行充分的测试，工具是必不可少的。到了 2002 年，Rick 和 Stefan 在《系统的软件测试》(*Systematic Software Testing*)一书中对软件测试做了进一步定义："测试是为了度量和提高被测软件的质量，对测试软件进行工程设计、实施和维护的整个生命周期过程"。这些经典论著对软件测试研究的理论化和体系化产生了巨大的影响。

近二十年来，随着计算机和软件技术的飞速发展，软件测试技术研究也取得了很大的突破，测试专家总结了很好的测试模型，例如著名的 V 模型、W 模型等，在测试过程改进方面提出了 TMM(Testing Maturity Model)的概念，在单元测试、自动化测试、负载压力测试以及测试管理等方面涌现了大量优秀的软件测试工具。

2．软件测试的国内外发展现状

在软件比较发达的国家，特别是美国，软件测试已经发展成为一个独立的产业，主要体现在以下几个方面。

(1) 软件测试在软件公司中占有重要的地位。比尔·盖茨曾在马萨诸塞州技术学院的一次演讲中说："在微软，一个典型的开发项目组中测试工程师要比编码工程师多得多，可以说我们花费在测试上的时间要比花费在编码上的时间多得多"。

(2) 软件测试理论研究蓬勃发展，每年举办各种各样的测试技术年会，发表了大量的软件测试研究论文，引领软件测试理论研究的国际潮流。

(3) 软件测试市场繁荣。美国有一些专业公司开发软件测试标准与测试工具，MI、Compuware、MaCabe、Rational 等都是著名的软件测试工具提供商，它们出品的测试工具已经占领了国际市场，目前我国使用的主流测试工具大部分是国外的产品，而且在世界各地都可以看到它们出品的软件测试工具，可见国外的软件测试已经形成了较大的产业。

在我国，软件测试可能算不上一个真正的产业，软件开发企业对软件测试认识淡薄，软件测试人员与软件开发人员往往比例失调，而在发达国家和地区软件测试已经成了一个产业。我们在软件测试实现方面并不比国外差，国际上优秀的测试工具，我们基本都有，这些工具所体现的思想我们也有深刻的理解，很多大型系统在国内都得到了很好的测试。

中国的软件测试技术研究起步于"六五"期间，主要是随着软件工程的研究而逐步发展起来的，由于起步较晚，与国际先进水平相比差距较大。直到 1990 年，成立了国家级的中国软件评测中心，测试服务才逐步开展起来。因此，我国无论是在软件测试理论研究还是在测试实践上，和国外发达国家都有不少的差距，主要体现在对软件产品化测试的技术研究还比较贫乏，从业人员较少，测试服务没有形成足够的规模等方面。但是，随着我国软件产业的蓬勃发展以及对软件质量的重视，软件测试也越来越被人们所看重，软件测试正在逐步成为一个新兴的产业。我国正在迈入测试时代，主要体现在以下几个方面。

(1) 我国著名的软件公司都已经或者正在建立独立的专职软件测试队伍，虽然测试人员规模以及所占比例还不能和国外的大公司相比，但是毕竟在公司内部贯彻了独立测试的意识。

(2) 国家人事部和信息产业部 2003 年关于职业资格认证第一次在我国有了"软件评测师"的称号，这是国家对软件测试职业的高度重视与认可。

(3) 在信息产业部关于计算机系统集成资质以及信息系统工程监理资质的认证中,软件测试能力已经被定为评价公司技术能力的一项重要指标。

(4) 2001 年,信息产业部发布的部长 5 号令,实行了软件产品登记认证制度,规定:凡是在我国境内销售的产品必须到信息产业部备案登记,而且要经过登记测试。

(5) 自 2001 年起,国家质检总局和信息产业部每年都通过测试对软件产品进行质量监督抽查。

(6) 国家各部委,各行业正在通过测试规范行业软件的健康发展,通过测试把不符合行业标准要求的软件挡在了门外,对行业信息化的健康发展起到了很好的促进作用。

(7) 用户对软件质量要求越来越高,信息系统验收不再走过场,而要通过第三方测试机构的严格测试来判定。

(8) "以测代评"正在成为我国科技项目择优支持的一项重要举措。例如,国家 863 计划对数据库管理系统、操作系统、办公软件、ERP 等项目的经费支持,都是通过第三方测试机构科学客观的测试结果来决定的。

(9) 软件测试正在成为部分软件学院的一门独立课程,对我国软件测试人才的培养起到了很好的作用。

(10) 第三方测试机构得到了蓬勃的发展。最近几年,在全国各地,新成立了多家软件测试机构,测试服务体系已经基本确立。

可见我国的软件测试行业正处于一个快速成长的阶段,我们有理由相信,经过一段时间的发展,我们会逐步缩小与国外发达国家的差距,从而带动整个软件产业的健康发展。

1.3 软件测试的概念解析

软件测试是为了发现错误而执行程序的过程。或者说,软件测试是根据软件开发各阶段的规格说明和程序内部结构而精心设计的一批测试用例(即输入数据及预期的输出结果),并利用这些测试用例去运行程序,以发现程序错误的过程。

1. 软件测试的定义

测试(Test)最早出现于古拉丁字,它有"罐"或"容器"的含义。在工业制造和生产中,测试被当作一个常规的检验产品质量的生产活动。测试的含义为"以检验产品是否满足需求为目标"。而软件测试活动包括很重要的任务,即发现错误。

"软件测试"的经典定义是在规定条件下对程序进行操作,以发现错误,对软件质量进行评估。

我们知道,软件是由文档、数据以及程序组成的,那么软件测试就应该是对软件形成过程的文档、数据以及程序进行的测试,而不仅是对程序进行的测试。

随着人们对软件工程化的重视以及软件规模的日益扩大,软件分析、设计的作用越来越突出。因此,做好软件需求和设计阶段的测试工作就显得非常重要。这就是我们提倡的测试概念扩大化,提倡软件全生命周期测试的理念。

软件测试就是在软件投入运行前,对软件需求分析、设计规格说明和编码的最终复审,是软件质量保证的关键步骤。通常对软件测试的定义有如下描述:软件测试是为了发现错

误而执行程序的过程。或者说,软件测试是根据软件开发各阶段的规格说明和程序的内部结构而精心设计一批测试用例,并利用这些测试用例去运行程序,以发现程序错误的过程。

软件测试是软件工程不可或缺的重要环节和重要组成部分,是用来确认一个程序的品质或性能是否符合开发之前所提出的一些要求。软件测试就是在软件投入运行前,对软件需求分析、设计规格说明和编码的最终复审,是软件质量保证的关键步骤与重要手段。软件测试的定义有许多种,其中比较权威的是国际标准IEEE在1983年提出的:“使用人工或自动手段来运行或测定某个系统的过程,其目的在于检验它是否满足规定的需求或是弄清预期结果与实际结果之间的差别”。软件测试是为了发现错误而执行程序的过程。软件测试在软件生存期中横跨两个阶段:通常在编写出每一个模块之后就对它做必要的测试(称为单元测试)。编码和单元测试属于软件生存期中的同一个阶段。在结束这个阶段后对软件系统还要进行各种综合测试,这是软件生存期的另一个独立阶段,即测试阶段。

针对软件的应用不同、规模不同、运行平台不同,需要选择不同的测试策略、测试方法,制定测试计划,编写测试用例,组织测试活动。例如,大型软件往往比小型软件需要进行更多的测试,并需要精心制定测试计划,有组织地执行测试活动;系统软件和支撑软件往往需要具有很好的兼容性、准确性和性能,为上层应用软件提供服务;基于Web的软件,往往负载能力成为决定其性能的核心指标之一;而嵌入式软件,需要严格控制其对运算能力和存储容量的需求,具有较高的性能。

2. 软件测试的目的

正确认识测试的目的是十分重要的,只有这样,才能设计出最能暴露错误的测试方案。测试的目的应从用户角度出发,通过软件测试暴露软件中潜在的错误和缺陷,而不是从软件开发者的角度出发,希望测试成为表明软件产品不存在错误,验证软件已正确实现用户的要求的过程。否则,开发者测试时会选择不易测试出错误和缺陷的用例,这与上述测试目的相违背。

软件测试的目的,第一是确认软件的质量,其一方面是确认软件做了所期望的事情(Do the right thing),另一方面是确认软件以正确的方式做了这个事件(Do it right)。

第二是提供信息,例如提供给开发人员或开发经理的反馈信息,为风险评估所准备的信息。

第三,软件测试不仅是在测试软件产品的本身,而且还包括软件开发的过程。如果一个软件产品开发完成之后发现了很多问题,这说明此软件开发过程很可能是有缺陷的。因此软件测试的第三个目的是保证整个软件开发过程是高质量的。

基于不同的立场,存在着两种完全不同的测试目的。从用户的角度出发,普遍希望通过软件测试暴露软件中隐藏的错误和缺陷,以考虑是否可以接受该产品。从软件开发者的角度出发,则希望成为表明软件产品中不存在错误的过程,验证该软件已正确地实现了用户的要求,确立人们对软件质量的信心。

测试的目标是能够以耗费最少时间与最小工作量找出软件系统中潜在的各种错误与缺陷。另外,我们应该认识到:测试只能证明程序中错误的存在,但不能证明程序中没有错误。因为即使实施了最严格的测试,仍然可能还有尚未被发现的错误或缺陷存在于程序中,因而测试不能证明程序没有错误,但可能查出程序中的错误。

软件测试人员的任务很清楚，就是站在使用者的角度上，通过不断地使用和攻击刚开发出来的软件产品尽量多地找出产品存在的问题，也就是我们所称的 Bug。

综上所述，软件测试的目的是以最少的测试用例集合测试出更多的程序中潜在错误。如何测试得彻底，怎样设计测试用例是测试的关键技术。简单地说，软件测试的目的就是通过寻找错误，尽可能地为修正错误提供更多的信息，从而保证软件系统的可用性。

3. 软件测试的基本原则

软件测试从不同的角度出发会派生出两种不同的测试原则，从用户的角度出发，就是希望通过软件测试能充分暴露软件中存在的问题和缺陷，从而考虑是否可以接受该产品；从开发者的角度出发，就是希望测试能表明软件产品不存在错误，已经正确地实现了用户的需求，确立人们对软件质量的信心。

人们为了提高测试的效率，在长期测试实验中积累了不少经验，下面列出了人们在实践中总结的主要基本原则。

(1) 尽早地并不断地进行软件测试。实际问题的复杂性、软件本身的复杂性与抽象性以及开发期各层人员工作的配合关系等各种错综复杂的因素使得软件开发的各个阶段都可能存在错误及潜在的缺陷。所以，软件开发的各阶段都应当进行测试。错误发现得越早，后阶段耗费的人力、财力就越少，软件质量相对就高一些。

(2) 程序员或程序设计机构应避免测试自己设计的程序。测试是为了找错，而程序员大多对自己所编的程序会有所偏袒，总认为自己编的程序问题不大或无错误存在，因此很难查出错误。此外，设计机构在测试自己程序时，由于开发周期和经费等问题的限制，要采用客观的态度是十分困难的。从工作效率来讲，最好由与原程序无关的程序员和程序设计机构进行测试。

(3) 测试用例中不仅要有输入数据，还要有与之对应的预期结果。测试前应当设定合理的测试用例。测试用例不仅要有输入数据，而且还要有与之对应的预期结果。如果在程序执行前无法确定预期的测试结果，由于人们的心理作用，可能把实际上是错误的结果当成是正确的。

(4) 测试用例的设计不仅要有合法的输入数据，还要有非法的输入数据。在设计测试用例时，不仅要有合法的输入测试用例，还要有非法的输入测试用例。在测试程序时，人们常忽视不合法的和预想不到的输入条件，倾向于考虑合法的和预期的输入条件。而在软件的实际使用过程中，由于各种因素的存在，用户可能会使用一些非法的输入，例如常会按错键或使用不合法的命令。对于一个功能较完善的软件来说，不仅当输入是合法的时候能正确运行，而且当有非法输入时，也应当能对非法的输入拒绝接受，同时给出对应的提示信息，使得软件便于使用。

(5) 在对程序修改之后要进行回归测试。在修改程序的同时时常又会引进新的错误，因而在对程序修改完之后，还应用以前的测试用例进行回归测试，有助于发现因修改程序而引进的新的错误。

(6) 程序中尚未发现的错误的数量通常与该程序中已发现的错误的数量成正比。经验表明：一段程序中若发现错误的数目越多，则此段程序中残存的错误数目也较多。例如，在美国的 IBM/370 的一个操作系统中，47%的错误(由用户发现的错误)仅与该系统的 4%的

程序模块有关。据此规律，在实际测验时，为了提高测试效率，要花较多的时间和代价来测试那些容易出错即出错多的程序段。而不要以为找到了几个错误，就认为问题已解决，不再需要继续测试了。

(7) 妥善保留测试计划、全部测试用例、出错统计和最终分析报告，并把它们作为软件的组成部分之一，为维护提供方便。设计测试用例要耗费相当大的工作量，若测试完随意丢弃，以后一旦程序改错后需重新测试时，将重复设计测试用例，这会造成很大的浪费，因而妥善保留与测试有关的资料，能为后期的维护工作带来方便。

(8) 应当对每一个测试结果做全面检查。这条重要的原则时常被人们忽视。不仔细、全面地检查测试结果，就会使得有错误征兆的输出结果漏掉。

(9) 严格执行测试计划，排除测试的随意性。测试计划内容应包括：所测软件的功能，输入和输出，测试内容，各项测试的进度安排，资源要求，测试资料，测试工具，测试用例的选择，测试的控制方式和过程，系统组装方式，跟踪规程，调试规程，回归测试的规定以及评价标准等。

4. 软件测试的对象

根据软件定义，软件包括程序、数据和文档。软件本身的含义对软件测试的指导意义在于，软件测试活动不应只局限于对程序的测试，也要充分考虑软件涉及的数据和描述软件的各相关文档。软件测试并不等于程序测试。软件测试应贯穿于整个软件生命周期中。在整个软件生命周期中，各阶段有不同的测试对象，形成了不同开发阶段的不同类型的测试。需求分析、概要设计、详细设计以及程序编码等各阶段所得到的文档，包括需求规格说明、概要设计规格说明、详细设计规格说明以及源程序，都应成为“软件测试”的对象。在软件编码结束后，对编写的每一个程序模块进行测试，称为“模块测试”或“单元测试”；在模块集成后，对集成在一起的模块组件，有时也可称为“部件”，进行测试，称为“集成测试”；在集成测试后，需要检测与证实软件是否满足软件需求说明书中规定的要求，这就称为“确认测试”。将整个程序模块集成为软件系统，安装在运行环境下，对硬件、网络、操作系统及支撑平台构成的整体系统进行测试，称为“系统测试”。

为了把握各个环节的正确性，人们需要进行各种验证和确认(Verification & Validation)工作。

验证(Verification)是保证软件正确实现特定功能的一系列活动和过程，目的是保证软件生命周期中的每一个阶段的成果满足上一个阶段所设定的目标。

确认(Validation)是保证软件满足用户需求的一系列的活动和过程，目的是在软件开发完成后保证软件与用户需求相符合。

验证与确认都属于软件测试，它包括对软件分析、设计以及程序的验证和确认。

在对需求理解与表达的正确性、设计与表达的正确性、实现的正确性以及运行的正确性的验证中，任何一个环节发生了问题都可能在软件测试中表现出来。

5. 软件测试的生命周期

图 1-2 给出了软件测试生命周期的模型，把测试的生命周期分为几个阶段。前三个阶段是引入程序错误阶段，也就是开发过程中的需求规格说明、设计、编码阶段，此时极易引入

错误或者导致开发过程中其他阶段产生错误。然后是通过测试发现错误的阶段，这需要通过使用一些适当的测试技术和方法来共同完成。后三个阶段是清除程序错误的阶段。其主要任务是进行缺陷分类、缺陷隔离和解决缺陷。其中在修复旧缺陷的时候很可能引进新的错误，导致原来能够正确执行的程序出现新的缺陷。

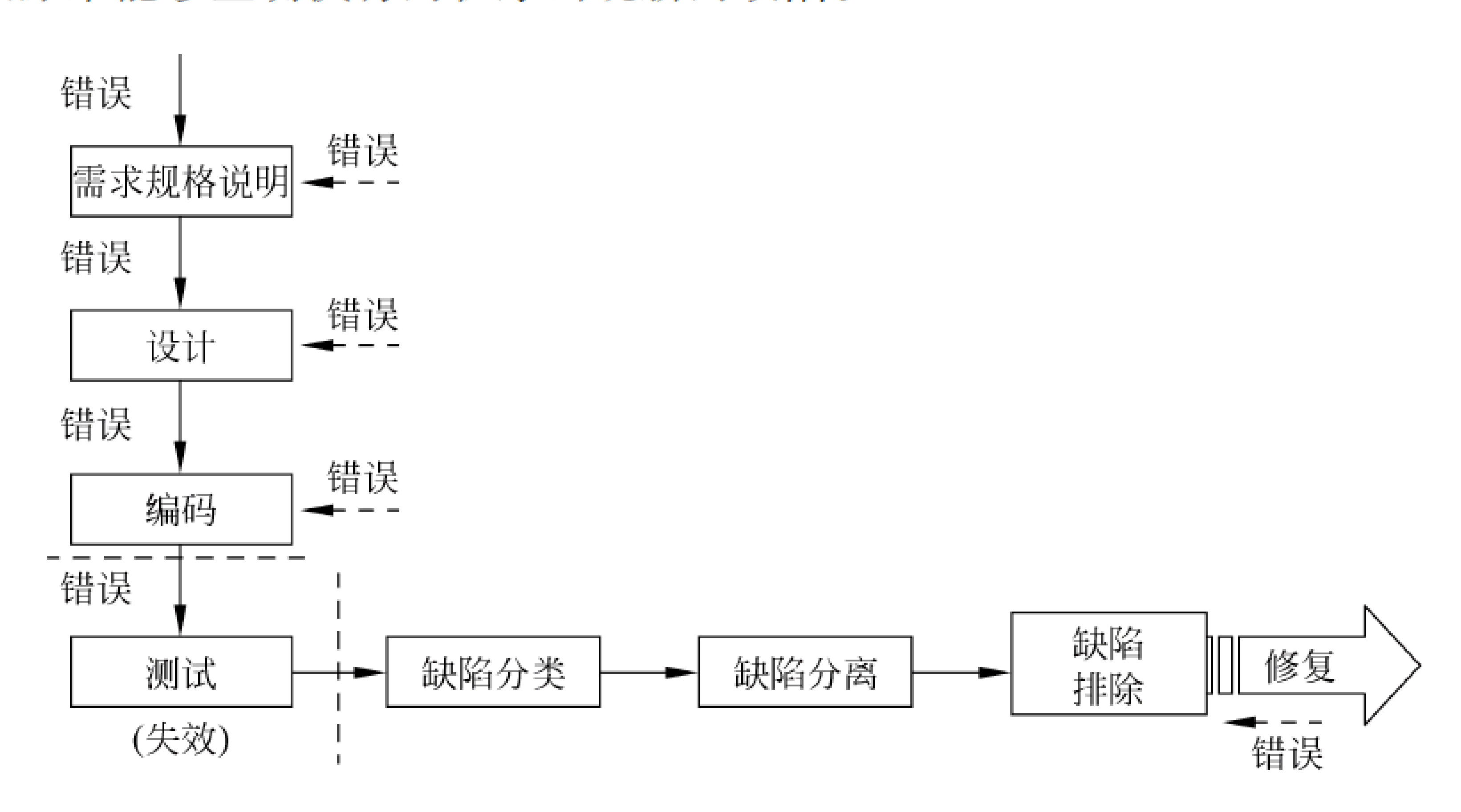


图 1-2　软件测试生命周期

在软件测试生命周期的每个阶段都要完成一些确定的任务，在执行每个阶段的任务时，可以采用行之有效的结构分析设计技术和适当的辅助工具；在结束每个阶段的任务时都进行严格的技术审查和管理复审。最后提交最终软件配置的一个或几个成分（文档或程序）。

6. 软件测试分类

按照全生命周期的软件测试概念，测试对象应该包括软件设计开发的各个阶段的内容。

1）按照开发阶段划分

按照开发阶段划分软件测试可分为单元测试、集成测试、确认测试、系统测试和验收测试。

（1）单元测试

单元测试又称模块测试，是针对软件设计的最小单位——程序模块进行正确性检验的测试工作。其目的在于检查每个程序单元能否正确实现详细设计说明中的模块功能、性能、接口和设计约束等要求，发现各模块内部可能存在的各种错误。单元测试需要从程序的内部结构出发设计测试用例。多个模块可以平行地独立进行单元测试。

（2）集成测试

集成测试也叫作组装测试。通常在单元测试的基础上，将所有的程序模块进行有序的、递增的测试。集成测试是检验程序单元或部件的接口关系，逐步集成为符合概要设计要求的程序部件或整个系统。

软件集成的过程是一个持续的过程，会形成很多个临时版本，在不断的集成过程中，功能集成的稳定性是真正的挑战。在每个版本提交时，都需要进行冒烟测试，即对程序主要功能进行验证。冒烟测试也叫版本验证测试、提交测试。

（3）确认测试

确认测试是通过检验和提供客观证据，证实软件是否满足特定预期用途的需求。确认

测试是检测与证实软件是否满足软件需求说明书中规定的要求。

(4) 系统测试

系统测试是为验证和确认系统是否达到其原始目标,而对集成的硬件和软件系统进行的测试。系统测试是在真实或模拟系统运行的环境下,检查完整的程序系统能否和系统(包括硬件、外设、网络和系统软件、支持平台等)正确配置、连接,并满足用户需求。

(5) 验收测试

按照项目任务书或合同、供需双方约定的验收依据文档进行的对整个系统的测试与评审,决定是否接受或拒收系统。

2) 按照测试实施组织划分

按照测试实施组织划分,软件测试可分为开发方测试、用户测试、第三方测试。

(1) 开发方测试

开发方测试通常也叫"验证测试"或"α 测试"。开发方通过检测和提供客观证据,证实软件的实现是否满足规定的需求。验证测试是在软件开发环境下,由开发者检测与证实软件的实现是否满足软件设计说明或软件需求说明的要求。主要是指在软件开发完成以后,开发方对要提交的软件进行全面的自我检查与验证,可以和软件的"系统测试"一并进行。

(2) 用户测试

在用户的应用环境下,用户通过运行和使用软件,检测与核实软件实现是否符合自己预期的要求。通常情况用户测试不是指用户的"验收测试",而是指用户的使用性测试,由用户找出软件的应用过程中发现的软件的缺陷与问题,并对使用质量进行评价。

β 测试通常被看成是一种"用户测试"。β 测试主要是把软件产品有计划地免费分发到目标市场,让用户大量使用,并评价、检查软件。通过用户各种方式的大量使用,来发现软件存在的问题与错误,把信息反馈给开发者修改。β 测试中厂商获取的信息,可以有助于软件产品的成功发布。

(3) 第三方测试

介于软件开发方和用户方之间的测试组织的测试。第三方测试也称为独立测试。软件质量工程强调开展独立验证和确认(IV&V)活动。IV&V 是由在技术、管理和财务上与开发组织具有规定程度独立的组织执行验证和确认过程。软件第三方测试也就是由在技术、管理和财务上与开发方和用户方相对独立的组织进行的软件测试。一般情况下是在模拟用户真实应用环境下,进行软件确认测试。

3) 按照测试技术划分

按照测试技术可分为白盒测试、黑盒测试、灰盒测试,也可分为静态测试和动态测试。静态测试是指不运行程序,通过人工对程序和文档进行分析与检查;静态测试技术又称为静态分析技术,静态测试实际上是对软件中的需求说明书、设计说明书、程序源代码等进行非运行的检查,静态测试包括走查、符号执行、需求确认等。动态测试是指通过人工或使用工具运行程序进行检查、分析程序的执行状态和程序的外部表现。这里讨论的白盒测试、黑盒测试、灰盒测试,在实现测试方法上既包括动态测试也包括静态测试。

(1) 白盒测试

通过对程序内部结构的分析、检测来寻找问题。白盒测试可以把程序看成装在一个透明的白盒子里,也就是清楚了解程序结构和处理过程,检查是否所有的结构及路径都是正确

的，检查软件内部动作是否按照设计说明的规定正常进行。白盒测试又称结构测试。

(2) 黑盒测试

通过软件的外部表现来发现其缺陷和错误。黑盒测试法把测试对象看成一个黑盒子，完全不考虑程序内部结构和处理过程。黑盒测试是在程序界面处进行测试，它只是检查程序是否按照需求规格说明书的规定正常实现。

(3) 灰盒测试

介于白盒测试和黑盒测试之间的测试。灰盒测试关注输出对于输入的正确性；同时也关注内部表现，但这种关注不像白盒测试那样详细、完整，只是通过一些表征性的现象、事件、标志来判断内部的运行状态。

灰盒测试结合了白盒测试和黑盒测试的要素。它考虑了用户端、特定的系统知识和操作环境。它在系统组件的协同性环境中评价应用软件的设计。

软件测试方法和技术的分类与软件开发过程相关联，它贯穿了整个软件生命周期。走查、单元测试、集成测试、系统测试用于整个开发过程中的不同阶段。开发文档和源程序可以应用单元测试应用走查的方法；单元测试可应用白盒测试方法；集成测试应用近似灰盒测试方法；而系统测试和确认测试应用黑盒测试方法。

1.4 软件测试的过程

软件测试一般按 4 个步骤进行，即单元测试、集成测试、确认测试和系统测试。图 1-3 显示出软件测试经历的 4 个步骤。单元测试集中对用源代码实现的每一个程序单元进行测试，检查各个程序模块是否正确地实现了规定的功能。然后，进行集成测试，根据设计规定的软件体系结构，把已测试过的模块组装起来，在组装过程中，检查程序结构组装的正确性。确认测试则是要检查已实现的软件是否满足了需求规格说明中确定的各种需求，以及软件配置是否完全、正确。最后是系统测试，把已经经过确认的软件纳入实际运行环境中，与其他系统成分组合在一起进行测试。严格地说，系统测试已超出了软件工程的范围。

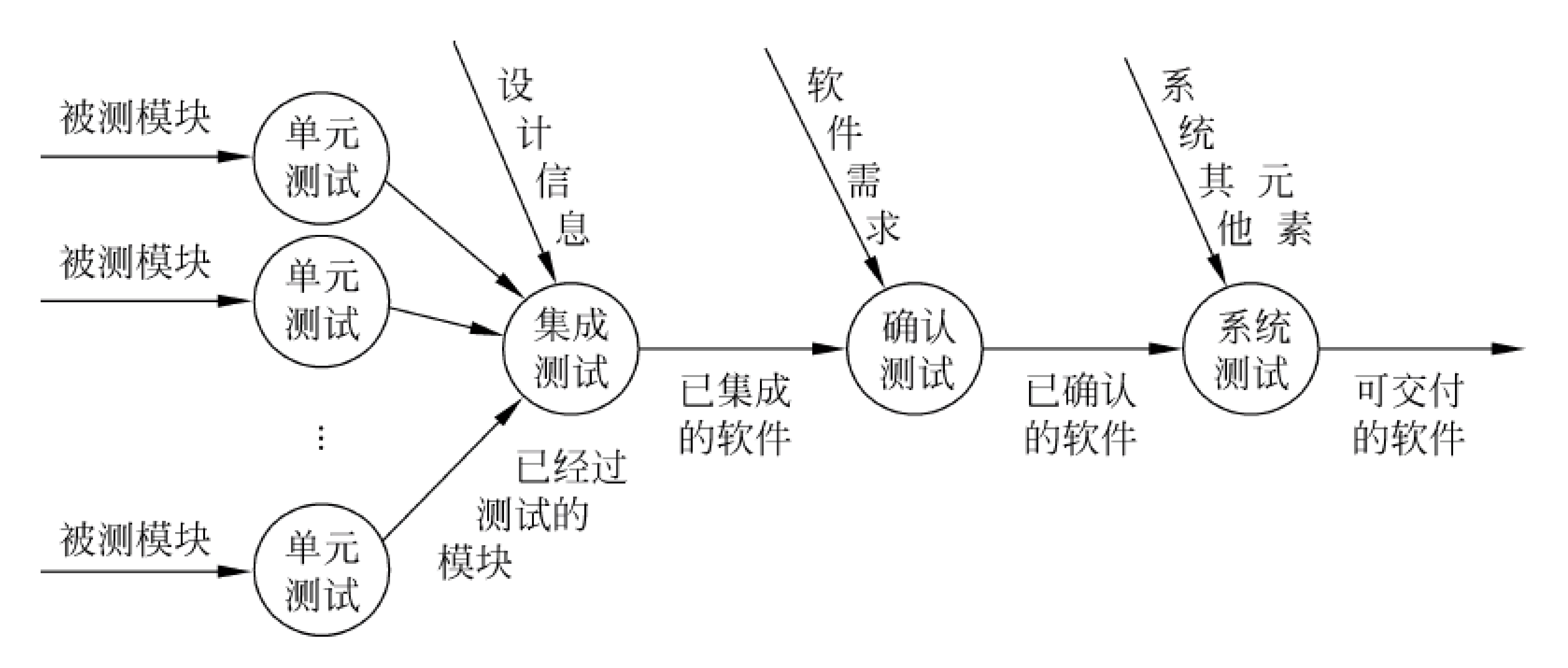


图 1-3 软件测试经历的步骤

1. 单元测试

单元测试是针对程序模块进行正确性检验的测试。其目的在于发现各模块内部可能存在的各种差错。单元测试需要从程序的内部结构出发设计测试用例。多个模块可以平行地

独立进行单元测试。

1）单元测试的内容

（1）模块接口测试：对通过被测模块的数据流进行测试。为此，对模块接口，包括参数表、调用子模块的参数、全程数据、文件输入输出操作都必须检查。

（2）局部数据结构测试：设计测试用例检查数据类型说明、初始化、默认值等方面的问题，还要查清全程数据对模块的影响。

（3）路径测试：选择适当的测试用例，对模块中重要的执行路径进行测试。对基本执行路径和循环进行测试可以发现大量的路径错误。

（4）错误处理测试：检查模块的错误处理功能是否包含错误或缺陷。例如，是否拒绝不合理的输入；出错的描述是否难以理解、是否对错误定位有误、是否出错原因报告有误、是否对错误条件的处理不正确；在对错误处理之前错误条件是否已经引起系统的干预等。

（5）边界测试：要特别注意数据流、控制流中刚好等于、大于或小于确定的比较值时出错的可能性。对这些地方要仔细地选择测试用例，认真加以测试。

此外，如果对模块运行时间有要求，还要专门进行关键路径测试，以确定最坏情况下和平均意义下影响模块运行时间的因素。这类信息对进行性能评价是十分有用的。

2）单元测试的步骤

通常单元测试在编码阶段进行。在源程序代码编制完成，经过评审和验证，确认没有语法错误之后，就开始进行单元测试的测试用例设计。利用设计文档，设计可以验证程序功能、找出程序错误的多个测试用例。对于每一组输入，应有预期的正确结果。

模块并不是一个独立的程序，在考虑测试模块时，同时要考虑它和外界的联系，用一些辅助模块去模拟与被测模块相联系的其他模块。这些辅助模块分为以下两种。

（1）驱动模块：相当于被测模块的主程序。它接收测试数据，把这些数据传送给被测模块，最后输出实测结果。

（2）桩模块：用于代替被测模块调用的子模块。桩模块可以做少量的数据操作，不需要把子模块所有功能都带进来，但不允许什么事情也不做。

被测模块、与它相关的驱动模块及桩模块共同构成了一个“测试环境”，见图 1-4。

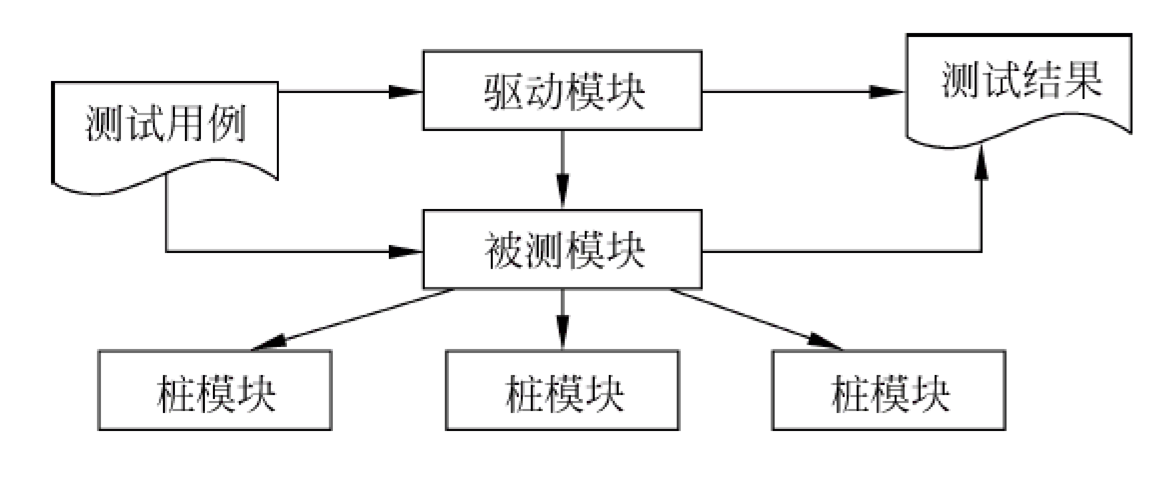


图 1-4　单元测试的步骤

如果一个模块要完成多种功能，且以程序包或对象类的形式出现，例如 Ada 中的包，MODULA 中的模块，C++ 中的类。这时可以将这个模块看成由几个小程序组成。对其中的每个小程序先进行单元测试要做的工作，对关键模块还要做性能测试。对支持某些标准规程的程序，更要着手进行互连测试。有人把这种情况特别称为模块测试，以区别单元测试。

2. 集成测试

在单元测试的基础上，需要将所有模块按照设计要求组装成系统。这时需要考虑：

(1) 在把各个模块连接起来的时候，穿越模块接口的数据是否会丢失；

(2) 一个模块的功能是否会对另一个模块的功能产生不利的影响；

(3) 各个子功能组合起来，能否达到预期要求的父功能；

(4) 全局数据结构是否有问题；

(5) 单个模块的误差累积起来，是否会放大，从而达到不能接受的程度；

(6) 单个模块的错误是否会导致数据库错误。

选择什么方式把模块组装起来形成一个可运行的系统，直接影响到模块测试用例的形式、所用测试工具的类型、模块编号的次序和测试的次序，以及生成测试用例的费用和调试的费用。通常，把模块组装成系统的方式有以下两种。

1) 一次性集成方式

它是一种非增殖式集成方式，也叫作整体拼装。使用这种方式，首先对每个模块分别进行模块测试，然后再把所有模块组装在一起进行测试，最终得到要求的软件系统。

由于程序中不可避免地存在涉及模块间接口、全局数据结构等方面的问题，所以一次试运行成功的可能性并不很大。

2) 增殖式集成方式

它又称渐增式集成方式。首先对一个个模块进行模块测试，然后将这些模块逐步组装成较大的系统，在组装的过程中边连接边测试，以发现连接过程中产生的问题。最后通过增殖逐步组装成为要求的软件系统。

(1) 自顶向下的增殖方式：将模块按系统程序结构，沿控制层次自顶向下进行集成。由于这种增殖方式在测试过程中较早地验证了主要的控制和判断点，在一个功能划分合理的程序结构中，判断常出现在较高的层次，较早就能遇到。如果主要控制有问题，尽早发现它能够减少以后的返工。

(2) 自底向上的增殖方式：从程序结构的最底层模块开始组装和测试。因为模块是自底向上进行组装，对于一个给定层次的模块，它的子模块(包括子模块的所有下属模块)已经组装并测试完成，所以不再需要桩模块。在模块的测试过程中需要从子模块得到的信息可以直接运行子模块得到。

(3) 混合增殖式测试：自顶向下增殖的方式和自底向上增殖的方式各有优缺点。自顶向下增殖方式的缺点是需要建立桩模块。要使桩模块能够模拟实际子模块的功能是十分困难的。同时涉及复杂算法和真正输入输出的模块一般在底层，它们是最容易出问题的模块，到组装和测试的后期才遇到这些模块，一旦发现问题，导致过多的回归测试。而自顶向下增殖方式的优点是能够较早地发现在主要控制方面的问题。自底向上增殖方式的缺点是“程序一直未能作为一个实体存在，直到最后一个模块加上去后才形成一个实体”。就是说，在自底向上组装和测试的过程中，对主要的控制直到最后才接触到。但这种方式的优点是不需要桩模块，而建立驱动模块一般比建立桩模块容易，同时由于涉及复杂算法和真正输入输出的模块最先得到组装和测试，可以把最容易出问题的部分在早期解决。此外，自底向上增殖的方式可以实施多个模块的并行测试。

有鉴于此，通常是把以上两种方式结合起来进行组装和测试。

(1) 衍变的自顶向下的增殖测试：它的基本思想是强化对输入输出模块和引入新算法模块的测试，并自底向上组装成为功能相当完整且相对独立的子系统，然后由主模块开始自顶向下进行增殖测试。

(2) 自底向上-自顶向下的增殖测试：它首先对含读操作的子系统自底向上直至根结点模块进行组装和测试，然后对含写操作的子系统做自顶向下的组装与测试。

(3) 回归测试：这种方式采取自顶向下的方式测试被修改的模块及其子模块，然后将这一部分视为子系统，再自底向上测试，以检查该子系统与其上级模块的接口是否适配。

3. 确认测试

确认测试又称有效性测试。它的任务是验证软件的有效性，即验证软件的功能和性能及其他特性是否与用户的要求一致。在软件需求规格说明书中描述了全部用户可见的软件属性，其中有一节叫作有效性准则，它包含的信息就是软件确认测试的基础。

在确认测试阶段需要做的工作如图 1-5 所示。首先要进行有效性测试以及软件配置复审，然后进行验收测试和安装测试，在通过了专家鉴定之后，才能成为可交付的软件。

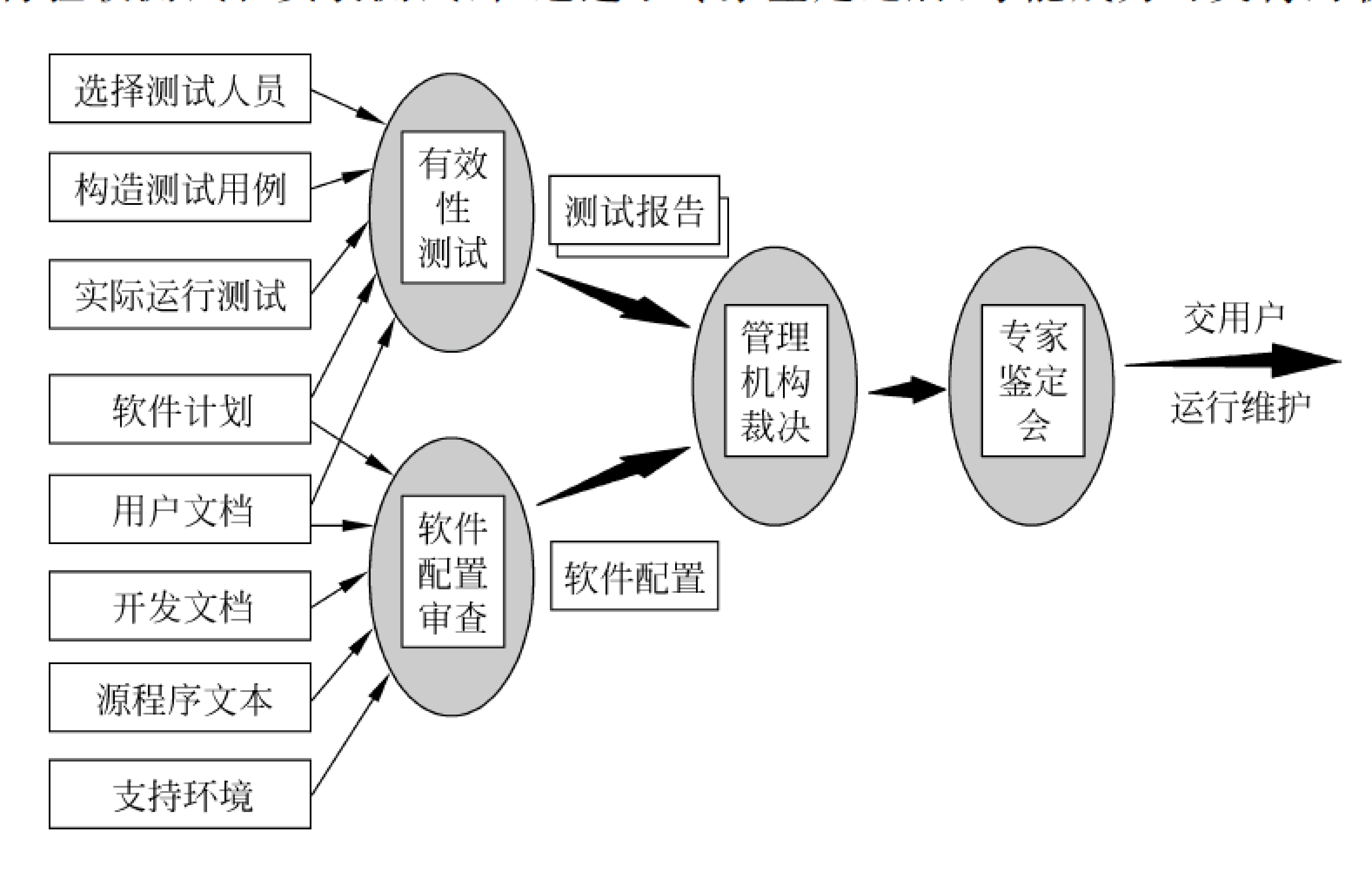


图 1-5 确认测试需做的工作

1) 进行有效性测试(功能测试)

有效性测试是在模拟的环境(可能就是开发的环境)下，运用黑盒测试的方法，验证被测软件是否满足需求规格说明书列出的需求。为此，需要首先制定测试计划，规定要做测试的种类。还需要制定一组测试步骤，描述具体的测试用例。通过实施预定的测试计划和测试步骤，确定软件的特性是否与需求相符，确保所有的软件功能需求都能得到满足，所有的软件性能需求都能达到，所有的文档都是正确且便于使用。同时，对其他软件需求，例如可移植性、兼容性、出错自动恢复、可维护性等，也都要进行测试，确认是否满足。

2) 软件配置复查

软件配置复查的目的是保证软件配置的所有成分都齐全，各方面的质量都符合要求，具有维护阶段所必需的细节，而且已经编排好分类的目录。

除了按合同规定的内容和要求,由人工审查软件配置之外,在确认测试的过程中,应当严格遵守用户手册和操作手册中规定的使用步骤,以便检查这些文档资料的完整性和正确性。必须仔细记录发现的遗漏和错误,并且适当地补充和改正。

3)验收测试

在通过了系统的有效性测试及软件配置审查之后,就应开始系统的验收测试。验收测试是以用户为主的测试。软件开发人员和QA(质量保证)人员也应参加。由用户参加设计测试用例,使用用户界面输入测试数据,并分析测试的输出结果。一般使用生产中的实际数据进行测试。在测试过程中,除了考虑软件的功能和性能外,还应对软件的可移植性、兼容性、可维护性、错误的恢复功能等进行确认。

4)α测试和β测试

在软件交付使用之后,用户将如何实际使用程序,对于开发者来说是无法预测的。因为用户在使用过程中常常会发生对使用方法的误解、异常的数据组合,以及产生对某些用户来说似乎是清晰的但对另一些用户来说却难以理解的输出等。

如果软件是为多个用户开发的产品的时候,让每个用户逐个执行正式的验收测试是不切实际的。很多软件产品生产者采用一种称为α测试和β测试的测试方法,以发现可能只有最终用户才能发现的错误。

α测试是由一个用户在开发环境下进行的测试,也可以是公司内部的用户在模拟实际操作环境下进行的测试。这是在受控制的环境下进行的测试。α测试的目的是评价软件产品的FURPS(即功能、可使用性、可靠性、性能和支持)。尤其注重产品的界面和特色。α测试人员是除产品开发人员之外首先见到产品的人,他们提出的功能和修改意见是特别有价值的。α测试可以从软件产品编码结束之时开始,或在模块(子系统)测试完成之后开始,也可以在确认测试过程中产品达到一定的稳定和可靠程度之后再开始。有关的手册(草稿)等应事先准备好。

β测试是由软件的多个用户在一个或多个用户的实际使用环境下进行的测试。与α测试不同的是,开发者通常不在测试现场。因而,β测试是在开发者无法控制的环境下进行的软件现场应用。在β测试中,由用户记下遇到的所有问题,包括真实的以及主观认定的,定期向开发者报告,开发者在综合用户的报告之后,做出修改,最后将软件产品交付给全体用户使用。β测试主要衡量产品的FURPS。着重于产品的支持性,包括文档、客户培训和支持产品生产能力。只有当α测试达到一定的可靠程度时,才能开始β测试。由于它处在整个测试的最后阶段,不能指望这时发现主要问题。同时,产品的所有手册文本也应该在此阶段完全定稿。由于β测试的主要目标是测试可支持性,所以β测试应尽可能由主持产品发行的人员来管理。

4. 系统测试

所谓系统测试,是将通过确认测试的软件,作为整个基于计算机系统的一个元素,与计算机硬件、外设、某些支持软件、数据和人员等其他系统元素结合在一起,在实际运行(使用)环境下,对计算机系统进行一系列的组装测试和确认测试。

系统测试的目的在于通过与系统的需求定义做比较,发现软件与系统定义不符合或与之矛盾的地方。系统测试的测试用例应根据需求分析规格说明来设计,并在实际使用环境

下运行。

每个具体测试的过程有三类输入：软件配置、测试配置和测试工具。软件配置包括软件需求说明书、设计说明书、源程序清单等文档。测试配置包括测试方案、测试计划、测试用例、测试驱动程序等文档。测试工具包括支持测试的软件。输出信息有修正软件的文件和预测可靠性或得出纠错后可交付使用的正确软件。测试的信息流是不断递归的过程，也是相对有限的测试过程，而不是无限的过程。软件测试的过程如图 1-6 所示。

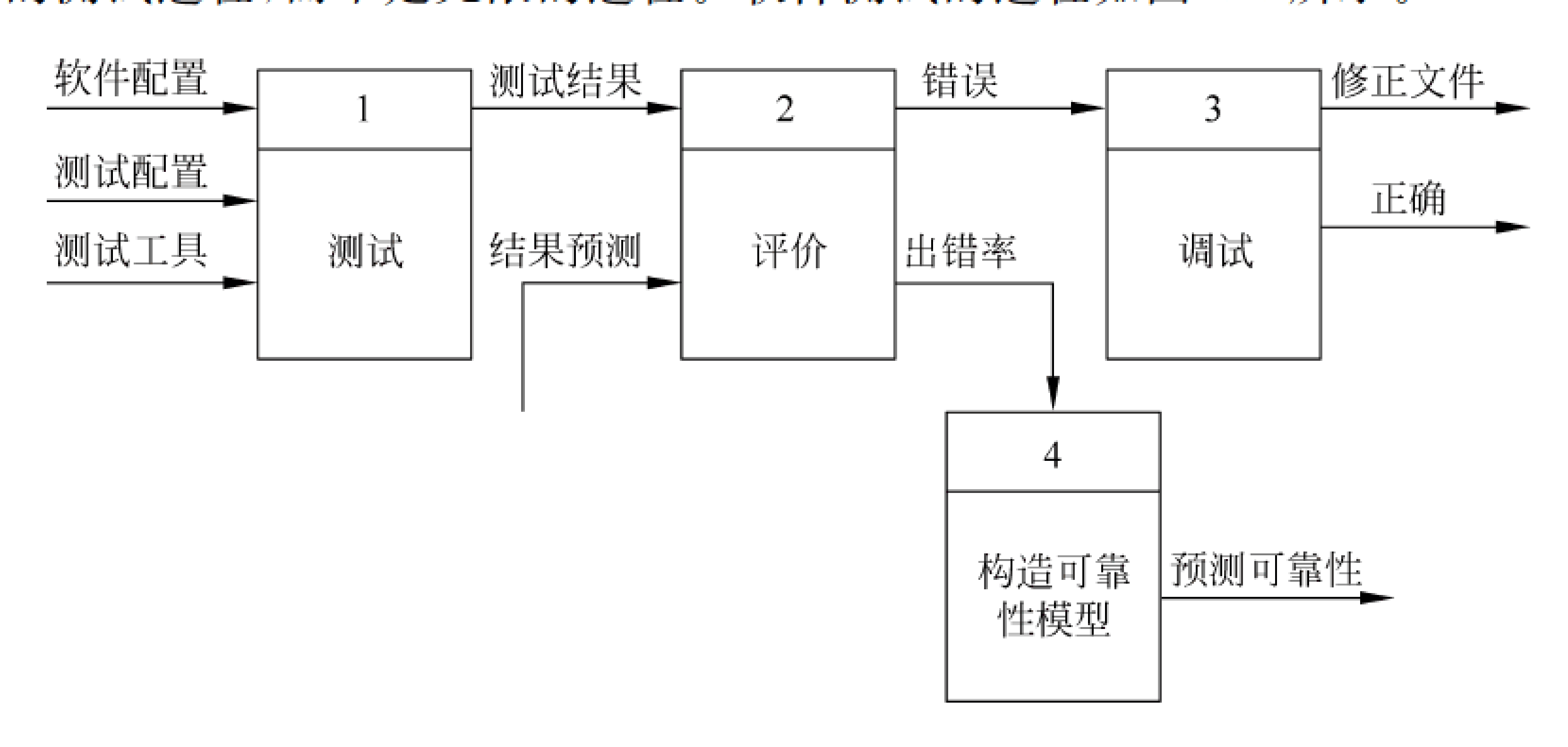


图 1-6 测试的过程

测试过程说明如下。

(1) 软件配置：指被测试软件的文件，如软件需求规格说明书、软件设计说明书和源程序清单等文档。

(2) 测试配置：指测试方案、测试计划、测试用例、测试驱动程序等文档。实际上，在整个软件工程过程中，测试配置只是软件配置的一个子集。

(3) 测试工具：是为了提高测试效率而设计的支持软件测试的软件。例如，测试数据自动生成程序、静态分析程序、动态分析程序、测试结果分析程序以及驱动测试的测试数据库等。

(4) 测试评价：由测试出的错误迹象，分析、找出错误的原因和位置，以便纠正和积累软件设计的经验。

(5) 纠错(调试)：是指找到出错的原因与位置并纠错，包括修正文件直到软件正确为止。纠错过程是测试过程中最无法预料的部分。为了诊断和纠正一个错误，可能需要一小时、一天、甚至几个月的时间。正是因为纠错本身所具有的不确定性，常常难以准确地安排测试日程表。

(6) 可靠性模型：通过对测试出的软件出错率的分析，建立模型，得出可靠的数据，指导软件的设计与维护。对测试结果进行收集和评价后，软件可靠性能够达到的质量指标也就清楚了。若出现一些有规律的、严重的、要求修改设计的错误，软件的质量和可靠性值得怀疑，应做进一步测试。另外，若软件功能看来完成得很好且遇到错误也容易纠正，从而可以得到两种不同的结论：一种是软件质量和可靠性是可以接受的；另一种是所进行的测试尚不足以发现严重的错误。若没有发现任何错误，可能是由于测试配置不够周到，依然有潜在的错误存在。若将错误放过，在维护阶段被用户发现时再纠正，所需费用将可能是开发阶段的 40～60 倍。

软件测试是一个极为复杂的过程。一个规范化的软件测试过程通常需包括以下基本的测试活动。

(1) 拟定软件测试计划;

(2) 编制软件测试大纲;

(3) 设计和生成测试用例;

(4) 实施测试;

(5) 生成软件测试报告。

实际上,软件测试过程与整个软件开发过程基本上是平行进行的。测试计划早在需求分析阶段就已经开始制定,其他相关工作,包括测试大纲的制定、测试数据的生成、测试工具的选择和开发等也应在测试阶段之前进行。充分的准备工作可以有效地克服测试的盲目性,缩短测试周期,提高测试效率,并且起到测试文档与开发文档互查的作用。

此外,软件测试的实施阶段是由一系列的测试周期(Test Cycle)组成的。在每个测试周期中,软件测试工程师将依据预先编制好的测试大纲和准备好的测试用例,对被测软件进行完整的测试。测试与纠错通常是反复交替进行的。当使用专业测试人员时,测试与纠错甚至是平行进行的,从而压缩总的开发时间。更重要的是,由于专业测试人员丰富的测试经验、所采用的系统化的测试方法、全时的投入,特别是独立于开发人员的思维,使得他们能够更有效地发现许多单靠开发人员很难发现的错误和问题。

软件测试大纲是软件测试的依据。它明确详尽地规定了在测试中针对系统的每一项功能或特性所必须完成的基本测试项目和测试完成的标准。无论是自动测试还是手动测试,都必须满足测试大纲的要求。

一般而言,测试用例是指为实施一次测试而向被测系统提供的输入数据、操作或各种环境设置。测试用例控制着软件测试的执行过程,它是对测试大纲中每个测试项目的进一步实例化。已有许多著名的论著总结了设计测试用例的各种规则和策略。从工程实践的角度讲有以下几条基本准则。

(1) 测试用例的代表性:能够代表各种合理和不合理的、合法的和非法的、边界和越界的,以及极限的输入数据、操作和环境设置等。

(2) 测试结果的可判定性:即测试执行结果的正确性是可判定的或可评估的。

(3) 测试结果的可再现性:即对同样的测试用例,系统的执行结果应当是相同的。

一般而言,软件测试从项目确立时就开始了,前后要经过以下一些主要环节:需求分析→测试计划→测试设计→测试环境搭建→测试执行→测试记录→缺陷管理→软件评估→RTM。

在进行有关问题阐述前,先明确分工,一般而言,需求分析、测试用例编写、测试环境搭建、测试执行等属于测试开发人员工作范畴,而测试执行以及缺陷提交等属于普通测试人员的工作范畴,测试负责人负责整个测试各个环节的跟踪、实施、管理等。

说明:

(1) 以上流程各环节并未包含软件测试过程的全部,如根据实际情况还可以实施一些测试计划评审、用例评审、测试培训等。在软件正式发行后,当遇到一些严重问题时,还需要进行一些后续维护测试等。

(2) 以上各环节并不是独立没联系的,实际工作千变万化,各环节一些交织、重叠在所

难免，例如编写测试用例的同时就可以进行测试环境的搭建工作，当然也可能由于一些需求不清楚而重新进行需求分析等。这就和我们国家提出建设有中国特色的社会主义国家一样，之所以有中国特色，那是因为国情不一样。所以在实际测试过程中也要做到具体问题具体分析，具体解决。

1. 需求分析

需求分析(Requiement Analyzing)应该说是软件测试的一个重要环节，测试开发人员对这一环节的理解程度如何将直接影响到接下来有关测试工作的开展。

可能有些人认为测试需求分析无关紧要，这种想法是很不对的。需求分析不但重要，而且至关重要。

一般而言，需求分析包括软件功能需求分析、测试环境需求分析、测试资源需求分析等。

其中最基本的是软件功能需求分析，测试一款软件首先要知道软件能实现哪些功能以及是怎样实现的。例如一款Smartphone包括VoIP、Wi-Fi以及Bluetooth等功能。那就应该知道软件是怎样来实现这些功能的，为了实现这些功能需要哪些测试设备以及如何搭建相应测试环境等，否则测试就无从谈起。

既然谈了需求分析，那么根据什么来分析呢？不能凭空设想。

总地说来，做测试需求分析的依据有软件需求文档、软件规格说明书以及开发人员的设计文档等，相信一些管理规范的公司在软件开发过程中都有这些文档。

2. 测试计划

测试计划(Test Plan)一般由测试负责人来编写。

测试计划的依据主要是项目开发计划和测试需求分析结果。测试计划一般包括以下一些方面。

1) 测试背景

(1) 软件项目介绍；

(2) 项目涉及人员(如软硬件项目负责人等)介绍以及相应联系方式等。

2) 测试依据

(1) 软件需求文档；

(2) 软件规格书；

(3) 软件设计文档；

(4) 其他，如参考产品等。

3) 测试资源

(1) 测试设备需求；

(2) 测试人员需求；

(3) 测试环境需求；

(4) 其他。

4) 测试策略

(1) 采取测试方法；

(2) 搭建哪些测试环境；

(3) 采取哪些测试工具以测试管理工具;

(4) 对测试人员进行培训等。

5) 测试日程

(1) 测试需求分析;

(2) 测试用例编写;

(3) 测试实施,根据项目计划,测试分成哪些测试阶段(如单元测试、集成测试、系统测试阶段,α、β 测试阶段等),每个阶段的工作重点以及投入资源等。

6) 其他

测试计划还要包括测试计划编写的日期、作者等信息,计划越详细越好。

计划赶不上变化,一份计划做得再好,当实际实施的时候就会发现往往很难按照原有计划开展。如在软件开发过程中资源匮乏、人员流动等都会对测试造成一定的影响。所以,这些就要求测试负责人能够从宏观上来调控了。在变化面前能够做到应对自如、处乱不惊是最好不过的了。

3. 测试设计

测试设计主要包括测试用例编写和测试场景设计两方面。

一份好的测试用例对测试有很好的指导作用,能够发现很多软件问题。

测试场景设计主要也就是测试环境问题。

4. 测试环境搭建

不同软件产品对测试环境有着不同的要求。如 C/S 及 B/S 架构相关的软件产品,那么对不同操作系统,如 Windows 系列、UNIX、Linux 甚至 iOS 等,这些测试环境都是必需的。而对于一些嵌入式软件,如手机软件,如果想测试一下有关功能模块的耗电情况、手机待机时间等,那么可能就需要搭建相应的电流测试环境了。当然测试中对于如手机网络等环境都有所要求。

测试环境很重要,符合要求的测试环境能够帮助我们准确地测出软件问题,并且做出正确的判断。

为了测试一款软件,可能根据不同的需求点要使用很多不同的测试环境。有些测试环境是可以搭建的,有些环境无法搭建或者搭建成本很高。不管如何,我们的目标是测试软件问题,保证软件质量。测试环境问题,应根据具体产品以及开发者的实际情况而采取最经济的方式。

5. 测试执行

测试执行过程又可以分为以下阶段:单元测试→集成测试→系统测试→出厂测试,其中每个阶段还有回归测试等。

从测试的角度而言,测试执行包括一个量和度的问题,也就是测试范围和测试程度的问题。例如一个版本需要测试哪些方面?每个方面要测试到什么程度?

从管理的角度而言,在有限的时间内,在人员有限甚至短缺的情况下,要考虑如何分工,如何合理地利用资源来开展测试。当然还要考虑以下问题。

(1) 当测试人员测试的执行不到位、敷衍了事时该如何解决？

(2) 测试效率问题，怎样提高测试效率？

(3) 根据版本的不同特点是只做验证测试还是采取冒烟测试抑或是系统全面测试？

(4) 当测试过程中遇到一些偶然性随机问题该怎样处理？

(5) 当版本中出现很多新问题时该怎样对待？测试停止标准？

⋮

总之，测试执行过程中会遇到很多复杂的问题，具体问题应具体解决。

6. 测试记录

缺陷记录总地说来包括两方面：由谁提交和缺陷描述。

一般而言，缺陷都是谁测试谁提交，当然有些公司可能为了保证所提交缺陷的质量，还会在提交前进行缺陷评估，以确保所提交的缺陷的准确性。

在缺陷的描述上，至少要包括以下一些方面的内容：

序号　标题　预置条件　操作步骤　预期结果　实际结果　注释　严重程度　概率　版本　测试者　测试日期

以上是描述一个 Bug 时通常所要描述的内容，当然在实际提交 Bug 时可以根据实际情况进行补充，如附上图片、Log 文件等。

另外，一个版本软件测试完毕，还要根据测试情况出份测试报告，这也是所要经过的一个环节。

7. 缺陷管理

缺陷管理方面，很多公司都采取缺陷管理工具来进行管理，常见缺陷管理工具有 Test Director、BugFree 等。

8. 软件评估

这里的评估指软件经过一轮又一轮测试后，确认软件无重大问题或者问题很少的情况下，对准备发给客户的软件进行评估，以确定是否能够发行给客户或投放市场。

软件评估小组一般由项目负责人、营销人员、部门经理等组成，也可能是由客户指定的第三方人员组成。

9. 测试总结

每个版本有每个版本的测试总结，每个阶段有每个阶段的测试总结，当项目完成 RTM 后，一般要对整个项目做个回顾总结，看有哪些做得不足的地方，有哪些经验可以对今后的测试工作做借鉴使用，等等。测试总结无严格格式、字数限制。应该说，测试总结还是很重要的。

10. 测试维护

由于测试的不完全性，当软件正式 release 后，客户在使用过程中，难免遇到一些问题，有的甚至是严重的问题，这就需要修改有关问题，修改后需要再次对软件进行测试、评估、发行。

在软件测试过程中，还需要注意以下几点。

(1) 测试人员参与需求分析和设计评审，确定需求的可测性，并贯穿到开发的整个过程；

(2) 项目组编写开发计划书(含集成计划)，测试人员据此产生创建计划书(或直接采用集成计划)；

(3) 测试人员细化测试计划和测试用例，产生测试计划书和测试用例说明书；

(4) 由项目组、SQA 人员、测试人员一起对测试计划书和测试用例说明书进行评审；

(5) 开发人员完成单元模块编码，然后对单元模块经过一系列静态检查和动态测试；

(6) 项目组执行集成测试，验证各通过单元测试的模块组合在一起的功能及其接口、数据传输的正确性，满足系统设计所规定的特性；

(7) 版本创建人员按集成或创建计划、从配置库中获得相应版本的源代码进行版本创建活动，并对创建版本进行管理；

(8) 测试人员对通过创建的工作产品执行冒烟测试，冒烟测试通过准则由测试人员和项目组事先在测试计划中约定，对冒烟测试未通过的系统，原则上由项目组当天解决问题，再次提交测试版本；

(9) 测试人员对完成集成的模块执行功能测试，即流程图所示功能集成测试。执行该过程实际上是对项目组集成测试的回归测试，它是增量式的；

(10) 重复步骤(5)～(9)，直至该版本所有功能都完成开发和经过功能集成测试；

(11) 测试人员根据测试计划中定义的系统测试策略，完成其他约定内容的测试，如性能测试、可使用性测试、安全性测试、安装/反安装测试等；

(12) 完成全部测试工作或根据时间驱动，测试负责人撰写测试分析报告；

(13) 测试分析报告由 SQA 人员负责组织评审，并由测试部经理批准；

(14) 对没达到测试出口准则的项目，由高级经理进行审批后，可做例外放行；

(15) 通过测试部测试的项目，在公司范围内进行产品版本发布并移交产品库。

1.5 软件测试与软件开发

1. 测试与软件开发各阶段的关系

软件开发过程是一个自顶向下，逐步细化的过程，首先在软件计划阶段定义了软件的作用域，然后进行软件需求分析，建立软件的数据域、功能和性能需求、约束和一些有效性准则。接着进入软件开发，首先是软件设计，然后再把设计用某种程序设计语言转换成程序代码。而测试过程则是依相反的顺序安排的自底向上，逐步集成的过程，低一级测试为上一级测试准备条件。此外还有两者平行地进行测试。

如图 1-7 所示，首先对每一个程序模块进行单元测试，消除程序模块内部在逻辑上和功能上的错误和缺陷。再对照软件设计进行集成测试，检测和排除子系统(或系统)结构上的错误。随后再对照需求，进行确认测试。最后从系统全体出发，运行系统，看是否满足要求。

2. 测试与开发的并行性

在软件的需求得到确认并通过评审后，概要设计工作和测试计划制定设计工作就要并

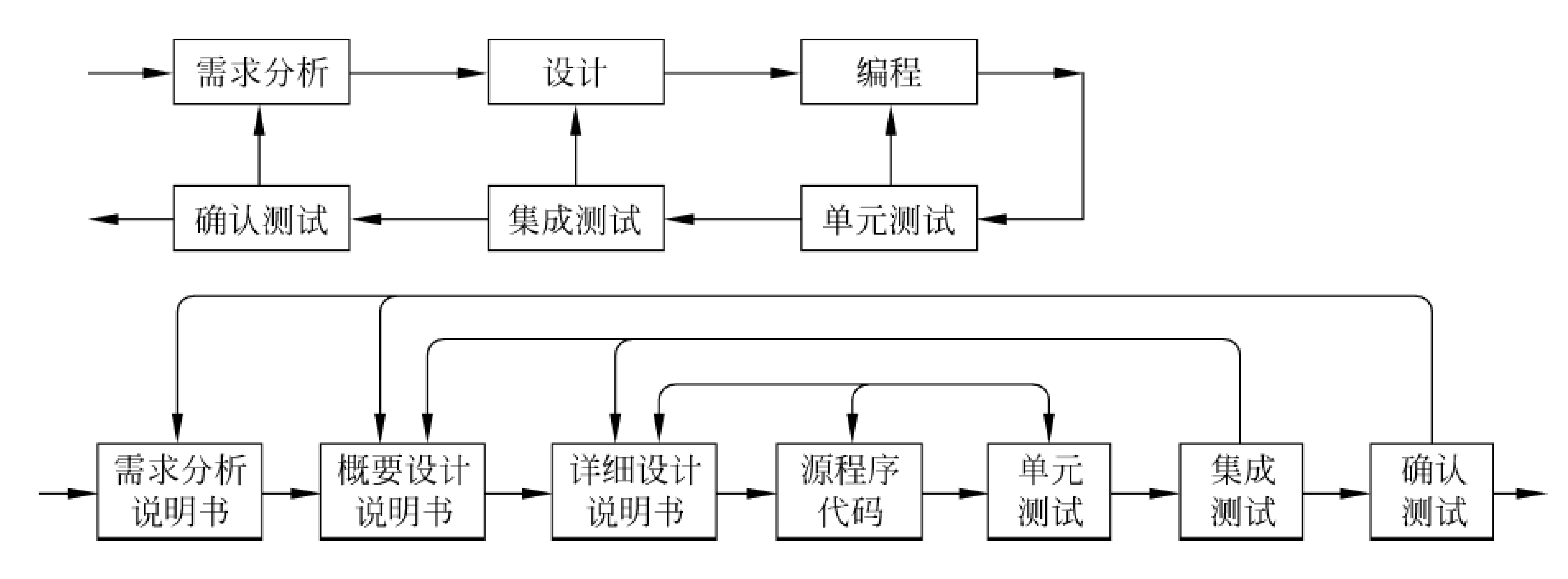


图 1-7　软件测试与软件开发过程的关系

行进行。如果系统模块已经建立，对各个模块的详细设计、编码、单元测试等工作又可并行。待每个模块完成后，可以进行集成测试、系统测试。并行流程如图 1-8 所示。

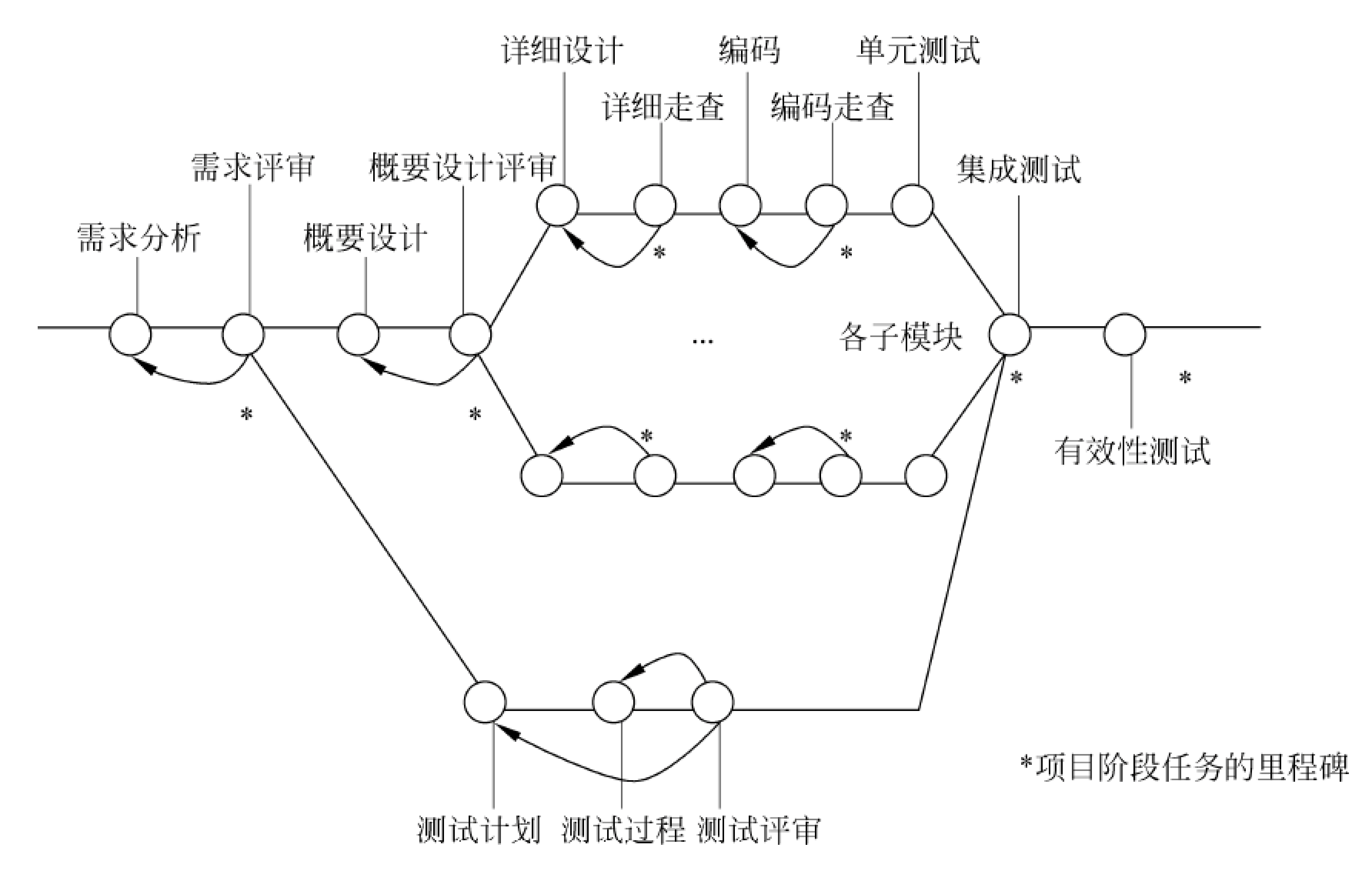


图 1-8　软件测试与软件开发的并行性

1.6　软件测试的重要性和实质

1. 软件测试的重要性

人们常常以为，开发一个程序是困难的，测试一个程序则比较容易。这其实是误解。设计测试用例是一项细致并需要高度技巧的工作，稍有不慎就会顾此失彼，发生不应有的疏漏。

不论是黑盒测试方法还是白盒测试方法，由于测试情况数量巨大，都不可能进行彻底的测试。所谓彻底测试，就是让被测程序在一切可能的输入情况下全部执行一遍。通常也称这种测试为“穷举测试”。“黑盒”法是穷举输入测试，只有把所有可能的输入都作为测试情况使用，才能以这种方法查出程序中所有的错误。实际上测试情况有无穷多个，人们不仅要测试所有合法的输入，而且还要对那些不合法但是可能的输入进行测试。“白盒”法是穷举

路径测试，贯穿程序的独立路径数是天文数字，但即使每条路径都测试了仍然可能有错误。第一，穷举路径测试决不能查出程序违反了设计规范，即程序本身是个错误的程序。第二，穷举路径测试不可能查出程序中因遗漏路径而出错。第三，穷举路径测试可能发现不了一些与数据相关的错误。E. W. Dijkstra 的一句名言对测试的不彻底性做了很好的注解："程序测试只能证明错误的存在，但不能证明错误不存在"。

在实际测试中，穷举测试工作量太大，实践上行不通，这就注定了一切实际测试都是不彻底的。当然就不能够保证被测试程序中不存在遗漏的错误。软件工程的总目标是充分利用有限的人力和物力资源，高效率、高质量地完成测试。为了降低测试成本，选择测试用例时应注意遵守"经济性"的原则。第一，要根据程序的重要性和一旦发生故障将造成的损失来确定它的测试等级；第二，要认真研究测试策略，以便能使用尽可能少的测试用例，发现尽可能多的程序错误。掌握好测试量是至关重要的，一位有经验的软件开发管理人员在谈到软件测试时曾这样说过："不充分的测试是愚蠢的，而过度的测试是一种罪孽"。测试不足意味着让用户承担隐藏错误带来的危险，过度测试则会浪费许多宝贵的资源。

测试是软件生存期中费用消耗最大的环节。测试费用除了测试的直接消耗外，还包括其他的相关费用。能够决定需要做多少次测试的主要影响因素如下。

（1）系统的目的。系统的目的的差别在很大程度上影响所需要进行的测试的数量。那些可能产生严重后果的系统必须要进行更多的测试。一台在 Boeing 757 上的系统应该比一个用于公共图书馆中检索资料的系统需要更多的测试。一个用来控制密封燃气管道的系统应该比一个与有毒爆炸物品无关的系统有更高的可信度。一个安全关键软件的开发组比一个游戏软件开发组要有苛刻得多的查找错误方面的要求。

（2）潜在的用户数量。一个系统的潜在用户数量也在很大程度上影响了测试必要性的程度。这主要是由于用户团体在经济方面的影响。一个在全世界范围内有几千个用户的系统肯定比一个只在办公室中运行的有两三个用户的系统需要更多的测试。如果不能使用，前一个系统的经济影响肯定比后一个系统大。除此以外，在分配处理错误的时候，所花的代价的差别也很大。如果在内部系统中发现了一个严重的错误，在处理错误的时候的费用就相对少一些，如果要处理一个遍布全世界的错误就需要花费相当大的财力和精力。

（3）信息的价值。在考虑测试的必要性时，还需要将系统中所包含的信息的价值考虑在内，一个支持许多家大银行或众多证券交易所的客户/服务器系统中含有经济价值非常高的内容。很显然，这一系统需要比一个支持鞋店的系统要进行更多的测试。这两个系统的用户都希望得到高质量、无错误的系统，但是前一种系统的影响比后一种要大得多。因此应该从经济方面考虑，投入与经济价值相对应的时间和金钱去进行测试。

（4）开发机构。一个没有标准和缺少经验的开发机构很可能开发出充满错误的系统。在一个建立了标准和有很多经验的开发机构中开发出来的系统中的错误不会很多，因此，对于不同的开发机构来说，所需要的测试的必要性也就截然不同。

然而，那些需要进行大幅度改善的机构反而不大可能认识到自身的弱点。那些需要更加严格的测试过程的机构往往是最不可能进行这一活动的，在许多情况下，机构的管理部门并不能真正地理解开发一个高质量的系统的好处。

（5）测试的时机。测试量会随时间的推移发生改变。在一个竞争很激烈的市场里，争取时间可能是制胜的关键，开始可能不会在测试上花多少时间，但几年后如果市场分配格局

已经建立起来了，那么产品的质量就变得更重要了，测试量就要加大。测试量应该针对合适的目标进行调整。

1）掌握好软件测试方法是提升软件企业工作质量的基础

（1）负载测试

负载测试指的是最常见的验证一般性能需求而进行的性能测试，也是用户最常见的性能需求，就是"既要马儿跑得快，又要马儿少吃草"。因此，负载测试主要是考察软件系统在既定负载下的性能表现。对负载测试可以有如下理解：首先，负载测试是站在用户的角度去观察在一定条件下软件系统的性能表现。其次，负载测试的预期结果是用户的性能需求得到满足。此指标一般体现为响应时间、交易容量、并发容量、资源使用率等。

（2）压力测试

压力测试是为了考察系统在极端条件下的表现，极端条件可以是超负荷的交易量和并发用户数。注意，这个极端条件并不一定是用户的性能需求，可能要远远高于用户的性能需求。可以这样理解，压力测试和负载测试不同的是，压力测试的预期结果就是系统出现问题，而我们要考察的是系统处理问题的方式。例如说，我们期待一个系统在面临压力的情况下能够保持稳定，处理速度可以变慢，但不能使系统崩溃。因此，压力测试是能让我们识别系统的弱点和在极限负载下程序将如何运行。例如，负载测试关心的是用户规则和需求，压力测试关心的是软件系统本身。对于它们的区别，可以用华山论剑的例子来更加形象地描述一下。如果把郭靖看作被测试对象，那么压力测试就像是郭靖和已经走火入魔的欧阳峰过招，欧阳锋蛮打乱来，毫无套路，尽可能地去打倒对方。郭靖要能应对住，并且不能丢进小命。而常规性能测试就好比郭靖和黄药师、洪七公三人约定，只要郭靖能分别接两位高手一百招，郭靖就算胜。至于一百招后哪怕郭靖会输掉那也不用管了。他只要能做到接下一百招，就算通过。

（3）并发测试

并发测试用于验证系统的并发处理能力。一般是和服务器端建立大量的并发连接，通过客户端的响应时间和服务器端的性能监测情况来判断系统是否达到了既定的并发能力指标。负载测试往往会使用并发来创造负载，之所以把并发测试单独提出来，是因为并发测试往往涉及服务器的并发容量，以及多进程/多线程协调同步可能带来的问题。这是要特别注意，必须测试的。

（4）基准测试

当软件系统中增加一个新的模块的时候，需要做基准测试，以判断新模块对整个软件系统的性能影响。按照基准测试的方法，需要打开/关闭新模块至少各做一次测试。关闭模块之前的系统各个性能指标记下来作为基准（Benchmark），然后与打开模块状态下的系统性能指标做比较，以判断模块对系统性能的影响。

（5）稳定性测试

"路遥知马力"，在这里要说的是和性能测试有关的稳定性测试，即测试系统在一定负载下运行长时间后是否会发生问题。软件系统的有些问题是不能一下子就暴露出来的，或者说是需要时间积累才能达到能够测试的程度。为什么会需要这样的测试呢？因为有些软件的问题只有在运行一天或一个星期甚至更长的时间才会暴露。这种问题一般是程序占用资源却不能及时释放而引起的。例如，内存泄漏问题就是经过一段时间积累才会慢慢变得显著，在运行初期却很难检测出来；还有客户端和服务器在负载运行一段时间后，建立了大量

的连接通路，却不能有效地复用或及时释放。

（6）可恢复测试

测试系统能否快速地从错误状态中恢复到正常状态。例如，在一个配有负载均衡的系统中，主机承受了压力无法正常工作后，备份机是否能够快速地接管负载。可恢复测试通常结合压力测试一起来做。每种测试有其存在的空间和目的。当接手一个软件项目后，在有限的资源条件下，选择去做哪一种测试，这应该根据当前软件过程阶段和项目的本身特点来做选择。例如，在集成测试的时候要做基准测试，在软件产品每个发布点要做性能测试。

2）编制好软件测试的过程是提升软件企业工作质量的前提

（1）拟定测试计划

在制定测试计划时，要推敲整个项目的开发时间和开发进度以及一些人为因素和客观条件等，使得测试计划是可行的。测试计划的内容主要有测试的内容、进度、测试所需的环境和条件、测试培训等。

（2）编制测试纲领

测试纲领是测试的依据。它清楚详尽地规定了在测试中针对系统的每一项功能或特性所必须完成的基础测试项目和测试完成的规范。

（3）依据测试纲领设计和生成测试用例

在设计测试用例的时候，可综合应用前面介绍的测试用例和设计技术，写作测试设计说明文档，其内容主要有被测项目、输入数据、测试过程、预期输出效果等。

（4）实施测试

测试的实施阶段是由一系列的测试周期组成的。在每个测试周期中，测试人员和开发人员将依据预先编制好的测试纲领和准备好的测试用例，对被测软件或设备进行完整的测试。

（5）生成测试报告

测试完成后，要形成相应的测试报告，主要对测试进行概要说明，列出测试的结论，指出缺陷。另外，给出一些建议，如可采用的修改方法，各项修改预计的工作量及修改的负责人员。

3）软件测试是提升软件企业硬件水平的根本手段

（1）软件测试是改进软件组织管理过程的必要途径。

软件测试过程是提取软件测试过程中可计量的属性，在测试过程进行中以一定频度不断地采集这些属性的值，并采用一些恰当的分析方法对得到的这些数据进行分析，从而量化地评定测试过程的能力和性能，提高测试过程的可视性，帮助软件组织管理以及改进软件测试过程。

评价：通过评价比较实际软件过程与标准或计划间的差异。过程评价是衡量过程好坏和过程改进效果的有效手段。控制：通过测试所反映的产品状态信息、项目状态信息和过程状态信息，可以帮助制定合理的管理控制措施，使产品偏离度、项目偏离度和过程偏离度在可控制的范围内，使项目过程的性能表现稳定，并且项目过程性能满足需求。预测：历史测试数据的积累能帮助预测当前项目的相关属性数据，有助于计划和决策的制定。改进：测试并不能直接改进过程，但基于测试的理解和评价为过程改进提供了有效的线索。根据这些线索再结合测试所记录的过程场景信息分析过程偏差的原因，帮助过程改进制定有效的变更措施。过程改进既是测试的结果，又是测试的动因。

（2）软件测试是保证软件质量的必要依据。

首先，软件质量是指软件的功能和性能满足用户需求和期望的程度，是软件的一种内在

特性。软件测试则是通过技术、流程、工具、人员以及管理手段,检测软件文档、软件中间产品和最终产品,查找和报告软件缺陷、错误以及隐患的技术。通过跟踪缺陷、错误及隐患的修正过程,确保软件产品、中间产品和文档符合软件工程过程需求和用户的最终需求。其次,软件测试就是利用测试工具按照测试方案和流程对产品进行功能和性能测试,甚至根据需要编写不同的测试工具,设计和维护测试系统,对测试方案可能出现的问题进行分析和评估。执行测试用例后,需要跟踪故障,以确保开发的产品适合需求。目前,国内软件企业对软件的质量越来越重视。很多软件厂商对软件开发过程进行了 CMM 及 ISO 的认证与标准贯彻。然而,要想保证软件质量,在注重软件开发过程规范化的同时,也不能忽视软件测试工作对于保证软件质量的重要意义。

总之,随着软件应用的领域不断深入,设计的复杂程度逐步扩大,开发的周期不断缩短,质量的要求不断提高,软件企业也面临着巨大挑战。因此,加强软件测试过程和技术,可以有效保证软件质量,这种观念正在被更多的软件企业人士理解、接受和实施,也是软件企业快速发展的必经之道。

2. 软件测试的实质

下面从以下几点了解软件测试的实质。

(1) 完全测试是不可能的。例如,完全的白盒测试和完全的黑盒测试都是不可能的。

① 白盒测试

图 1-9 是一个不超过 100 行的程序结构图,有大概 100 000 000 000 000 条可能的执行路径。以每秒执行 1000 个测试用例的速度计算,完成所有可能路径的测试大概需要 3170 年。

② 黑盒测试

如图 1-10 所示,在 32 位的计算机上运行,只考虑 x,y 是整数,不同的测试数据组合最大可能数目为:2^{64},以每秒执行 1000 个测试用例的速度计算,完成测试大概需要工作 5 亿年。

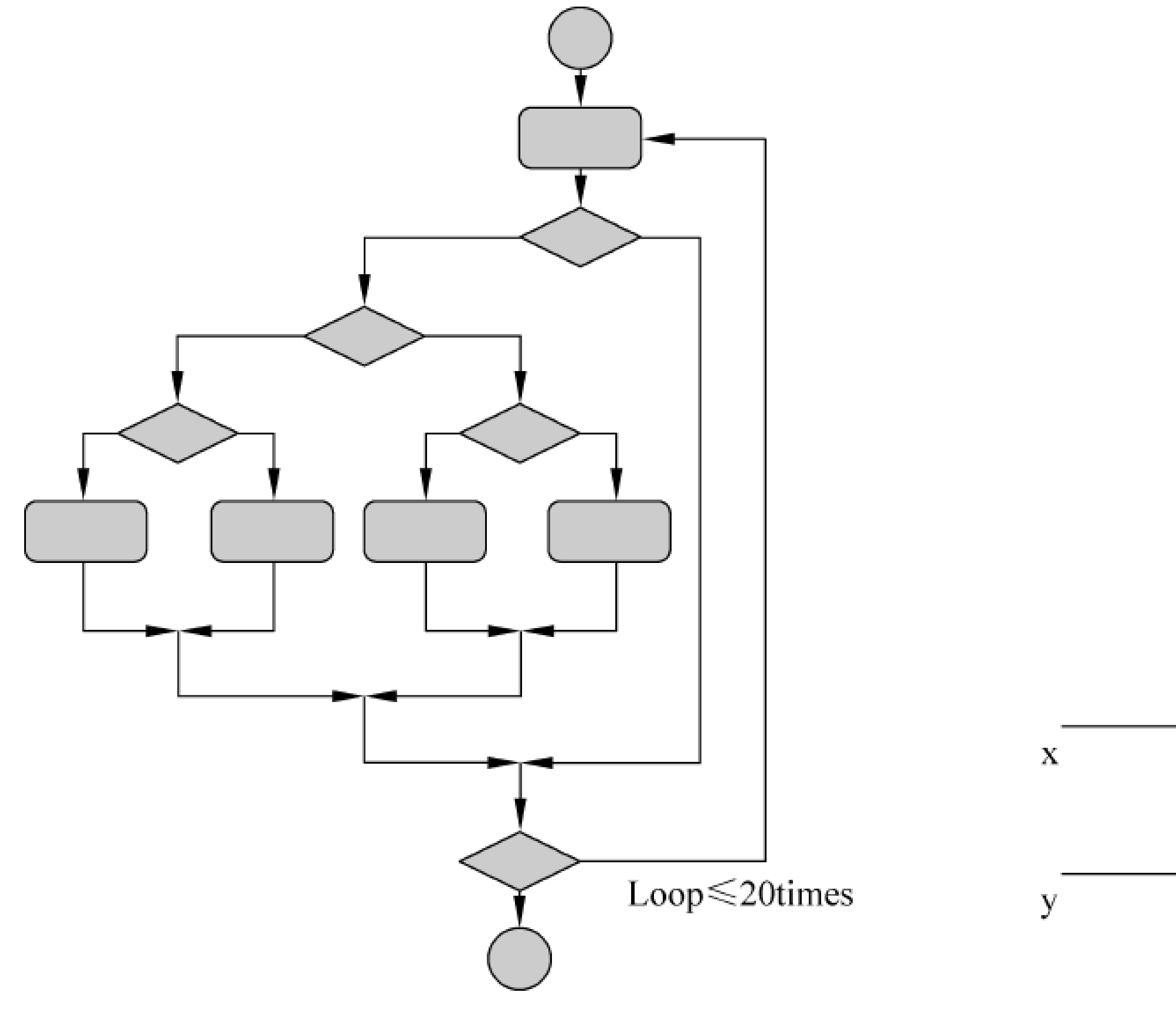


图 1-9 不超过 100 行的程序结构图

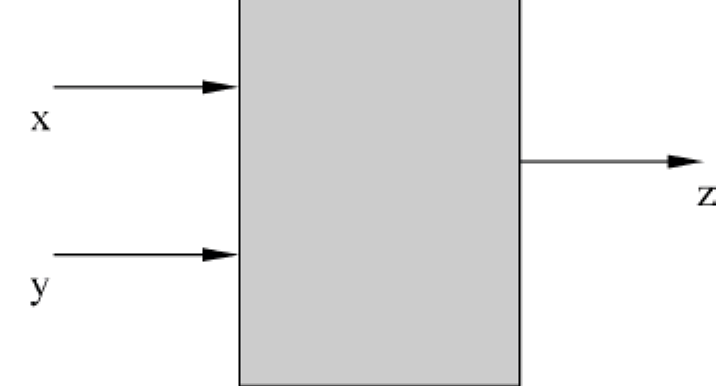


图 1-10 不同的测试数据组合

无法进行完全测试的原因有以下几点。

① 输入量太大；

② 输出结果太多；

③ 软件执行路径太多；

④ 软件说明书本身就是主观的产物，可能不正确，也经常会变动。

(2) 软件测试是有风险的行为。

(3) 测试很难显示潜伏的软件缺陷。

(4) 找到的软件缺陷越多，说明软件缺陷越多。

(5) 杀虫剂现象：软件测试越多，软件的免疫力越强。

(6) 并非所有的软件缺陷都要修复。

(7) 难以说清的软件缺陷。

至少满足以下 5 个规则之一才称为软件缺陷。

① 软件未实现产品说明书要求的功能。

② 软件出现了产品说明书指明不应出现的错误。

③ 软件实现了产品说明书未提到的功能。

④ 软件未实现产品说明书虽未明确提及但应该实现的目标。

⑤ 软件难以理解、不易使用、运行缓慢或者从测试员的角度看最终用户会认为不好。

(8) 产品说明书不断变化，没有最终版本。

(9) 软件测试员在产品小组中不受欢迎。

(10) 软件测试是一项讲究条理的技术工作。

思考题

1. 举例说明什么是软件缺陷。
2. 软件测试技术的发展历史和现状如何？
3. 软件测试一般需要经历哪几个步骤？
4. 软件测试一般所需要经历的过程是怎样的？
5. 软件测试与软件开发的关系是怎样的？
6. 软件测试的实质是什么？

第2章 软件质量

本章学习重点

- 理解质量与软件质量的定义。
- 熟悉常见的软件质量模型。
- 了解标准的发展情况。
- 熟悉软件质量与软件测试、软件质量保证与软件测试之间的联系。

本章学习难点

熟悉常见的软件质量模型。

2.1 质量的定义

想提高质量，就必须给出质量定义并测量它。大众化的观点和含糊其词的定义，不利于质量工程的开展。质量是多维的概念，包括实体、实体的属性和对实体的观点。

早在1970年，Juran和Gryna把质量定义为“适于使用”。1979年，Crosby将质量定义为“符合需求”。“符合需求”隐含着需求必须明确地说明的意思；而“适于使用”隐含顾客的需求和预期。顾客的需求和预期包括产品和服务是否适合顾客使用。

国际标准ISO 8404《质量管理与质量保证术语》对质量的定义是“反映实体满足明确的和隐含的需要的能力的特性的总和”。作为软件质量，在国际标准ISO 14598《软件工程产品评价》中也是这样定义的：“实体特性的总和，满足明确或隐含要求的能力”。

1. QA和QC

QA即英文Quality Assurance的简称，中文意思是质量保证。

QC即英文Quality Control的简称，中文意思是质量控制。

QC和QA的主要区别：前者是保证产品质量符合规定，后者是建立体系并确保体系按要求运作，以提供内外部的信任。同时，QC和QA又有相同点：即QC和QA都要进行验证，如QC按标准检测产品就是验证产品是否符合规定要求，QA进行内审就是验证体系运

作是否符合标准要求,又如 QA 进行出货稽核和可靠性检测,就是验证产品是否已按规定进行各项活动,是否能满足规定要求,以确保工厂交付的产品都是合格和符合相关规定的。

2. 测度与度量

一个实体的质量好坏是需要测量的,而测试就需要首先建立质量指标体系或质量模型,然后使用特定测量方法才能实施测量。测度的运用是建立测量方法的依据,也是解决软件质量测量的关键。

测度是把数字和符号分配给现实世界实体的属性,根据明确定义规则来定义它们 Fenton(FEN91)。软件工程中使用的“独立”有多种含义,在软件质量中用于测量的一种量化的标度和方法即为“测度”,而名词“度量”用来指测量的结果。

2.2 软件质量

信息技术的飞速发展,使软件产品应用到社会的各个领域,软件产品的质量自然成为人们共同关注的焦点.不论软件的生产者还是软件的使用者,均生存在竞争的环境中,软件开发商为了占有市场,必须把软件质量作为企业的重要目标之一,以免在激烈的竞争中被淘汰出局。用户为了保证自己业务的顺利完成,当然希望选用优质的软件。质量不佳的软件产品不仅会使开发商的维护费用和用户的使用成本大幅增加,还可能产生其他的责任风险。一些关键的应用中,如民航订票系统、银行结算系统、证券交易系统、自动飞行控制软件、军事防御和核电站安全控制系统等,若使用质量有问题的软件,会造成灾难性的后果。

软件测试是软件质量保证工作的一个重要环节。因此,首先应对什么是软件质量有一个明确的认识。

在 1991 年,软件产品质量评价国际标准 ISO 9126 中定义的软件质量是:“软件满足规定或潜在用户需求特性的总和”。到 1999 年,软件产品质量评价国际标准 ISO 14598 把软件质量定义为:“软件特性的总和,软件满足规定或潜在用户需求的能力”。

把“能力”加到“质量”的定义中,是由于一般人对“质量”的理解是一个实体的“属性”,“属性”好就是质量好。但属性是内在的特性,内在的特性好,不一定能胜任和完成用户的任务。在定义中明确“满足规定或潜在用户需求的能力”,体现了软件质量也是关于软件特性具备的“能力”。在 2001 年,软件产品质量评价国际标准 ISO 9126 中定义的软件质量包括“内部质量”、“外部质量”和“使用质量”三部分。也就是说“软件满足规定或潜在用户需求的能力”要从软件在内部、外部和使用中的表现来衡量。

为满足软件的各项规定的或隐含的功能、性能需求,符合文档化开发标准,就需要相应地设计出一些质量特性及其组合,作为在软件开发与维护中的重要考虑因素。如果这些质量特性及其组合都能在产品中得到满足,则这个软件产品的质量就是高的。这些被定义出来的特性及其组合就称为软件的“质量目标”。

从上述软件质量的定义中,反映出了以下三个方面的问题。

(1) 软件需求是度量软件质量的基础。不符合需求的软件就不具备质量。

(2) 软件人员必须遵循软件过程规范,用工程化的方法来开发软件。如果不遵守这些规程,软件质量就没有保证。

(3) 往往会有一些隐含的需求没有明确地提出来。如果软件只是满足那些规定的需求而不可能满足那些可能存在的隐含需求,软件质量也不能保证。

软件质量是指与软件产品满足规定的和隐含的需求的能力有关的特征和特性的全体。概括地说,软件质量就是“软件与明确的和隐含的定义的需求相一致的程度”。具体地说,软件质量是软件符合明确叙述的功能和性能需求、文档中明确描述的开发标准以及所有专业开发的软件都应具有的隐含特征的程度。

软件需求是度量软件质量的基础,与需求不一致就是质量不高。指定的标准定义了一组指导软件开发的准则,如果没有遵守这些准则,几乎肯定会导致质量不高。通常,有一组没有显式描述的隐含需求(如期望软件是容易维护的)。如果软件满足明确描述的需求,但却不满足隐含的需求,那么软件的质量仍然是值得怀疑的。

软件质量是一个软件企业成功的必要条件,其重要性无论怎样强调都不过分。软件质量与传统意义上的质量概念并无本质差别,只是针对软件的某些特性进行了调整。

软件危机曾经是软件界甚至整个计算机界最热门的话题。为了解决这场危机,软件从业人员、专家和学者做出了大量的努力。现在人们已经逐步认识到所谓的软件危机实际上仅是一种状况,那就是软件中有错误,正是这些错误导致了软件开发在成本、进度和质量上的失控。有错是软件的属性,而且是无法改变的,因为软件是由人来完成的,所有由人做的工作都不会是完美无缺的。问题在于我们如何去避免错误的产生和消除已经产生的错误,使程序中的错误密度达到尽可能低的程度。

软件测试在软件生命周期中占据重要的地位,在传统的瀑布模型中,软件测试学仅处于运行维护阶段之前,是软件产品交付用户使用之前保证软件质量的重要手段。近年来,软件工程界趋向于一种新的观点,即认为软件生命周期每一阶段中都应包含测试,从而检验本阶段的成果是否接近预期的目标,尽可能早地发现错误并加以修正,如果不在早期阶段进行测试,错误的延时扩散常常会导致最后成品测试的巨大困难。

1. 影响软件质量的主要因素

影响软件质量的主要因素是从管理角度对软件质量的度量。可划分为三组:开发软件的组织、开发过程、开发过程所使用的方法和技术,分别反映用户在使用软件产品时的三种观点:正确性、健壮性、效率、完整性、可用性、风险(产品运行);可理解性、可维修性、灵活性、可测试性(产品修改);可移植性、可再用性、互运行性(产品转移)。

(1) 性能(Performance)是指系统的响应能力,即要经过多长时间才能对某个事件做出响应,或者在某段时间内系统所能处理的事件个数。在指定条件下,用软件实现某种功能所需的计算机资源(包括时间)的有效程度。

(2) 可用性(Availability)是指系统能够正常运行的时间比例。

(3) 可靠性(Reliability)是指系统在应用或者错误面前,在意外或者错误使用的情况下维持软件系统功能特性的能力。在规定的时间和条件下,软件所能维持其性能水平的程度。

(4) 健壮性(Robustness)是指在处理或者环境中系统能够承受的压力或者变更能力。

(5) 安全性(Security)是指系统向合法用户提供服务的同时能够阻止非授权用户使用的企图或者拒绝服务的能力。

(6) 可修改性(Modification)是指能够快速地以较高的性能价格比对系统进行变更的

能力。在一个运行软件中，当环境改变或软件发生错误时，进行相应修改所做努力的程度。

(7) 可变性(Changeability)是指体系结构扩充或者变更成为新体系结构的能力。软件从一个计算机系统或环境移植到另一个系统或环境的容易程度。

(8) 易用性(Usability)是衡量用户使用软件产品完成指定任务的难易程度。对于一个软件，用户学习、操作、准备输入和理解输出所做努力的程度。

(9) 可测试性(Testability)是指软件发现故障并隔离定位其故障的能力特性，以及在一定的时间或者成本前提下进行测试设计、测试执行能力。

(10) 功能性(Function Ability)是指系统所能完成所期望工作的能力。软件所实现的功能达到它的设计规范和满足用户需求的程度。

(11) 互操作性(Inter-Operation)是指系统与外界或系统与系统之间的相互作用能力。

2. 软件开发

1) 需求分析

确保客户所要求的系统是可行的。

确保客户指定的需求确实能够满足他的真正要求。

避免开发者和客户之间的误解。

向用户提供为满足他所提出的需求而实际构建的适当软件系统。

2) 软件规格说明

通过建立需求跟踪文档，确保规格说明书与系统需求保持一致。

确保规格说明书能适当地改进系统的灵活性、可维护性以及性能。

确保已建立了测试策略。

确保已建立了现实的开发进度表，包括预定的评审。

确保已为系统设计了正式的变更规程。

3) 设计

确保已建立用于描述设计的标准，并且确保遵循这些标准。

确保适当地控制并用文档记录对设计进行的变更。

确保在系统设计组件已按照商定的准则得到批准之后才开始编码。

确保对设计的评审按照进度进行。

确保代码遵循已建立的风格、结构和文档标准。

确保代码经过适当测试和集成，同时对编码模块的修改得到适当的标识。

查看代码编写是否遵循既定的进度。

确保代码评审按照进度进行。

4) 测试

确保测试计划的建立和遵循。

确保创建的测试计划能够满足所有系统规格说明书的要求。

确保经过测试和返工后软件与规格说明书保持一致。

5) 维护

确保代码和文档的一致性。

确保对已建立的变更控制过程进行监测，包括将变更集成到软件的产品版本中的过程。

确保对代码的修改遵循编码标准，并且要对其进行评审，不要破坏整个代码结构。

2.3　软件质量模型

从测量的角度看，影响软件质量的因素可以分为两大类：可直接测量（如每个功能点的错误）和间接度量（如可用性、可维护性）。每种类型的测度都必须发生。

软件质量模型定义一个软件的质量，等价于为该软件定义一系列质量特性。通常用软件质量模型来描述影响软件质量的特性。

目前，软件质量模型的研究主要分为两个方向：一是根据经验提出软件质量模型；二是给出一种构建软件质量模型的方法。前者主要是建立各种通用的质量模型来描述软件或信息系统，从而对软件的质量进行预测、度量以及评估。后者主要是提供了如何建立质量模型的方法，包括如何建立质量属性之间的联系以及如何分析质量属性等。

软件质量模型分为两类：层次模型和关系模型。其中，比较著名的层次模型包括McCall模型、Boehm模型和ISO 9126质量模型。这些质量模型用来描述软件或者信息系统，从而可以对软件质量进行预测、度量以及评估，最终在软件开发的整个生命周期，实施有效的质量保证的措施，确保这些软件质量特性得以满足。

1. McCall 模型

早期的软件质量模型是1977年McCall和他的同事建立的，提出了影响质量因素的有用分类。McCall质量模型集中在软件产品的三个重要方面：操作特性（产品运行）、承受可改变能力（产品修订）、新环境适应能力（产品变迁）。

McCall模型是最早的质量模型之一，属于层次模型的类型，见图2-1。

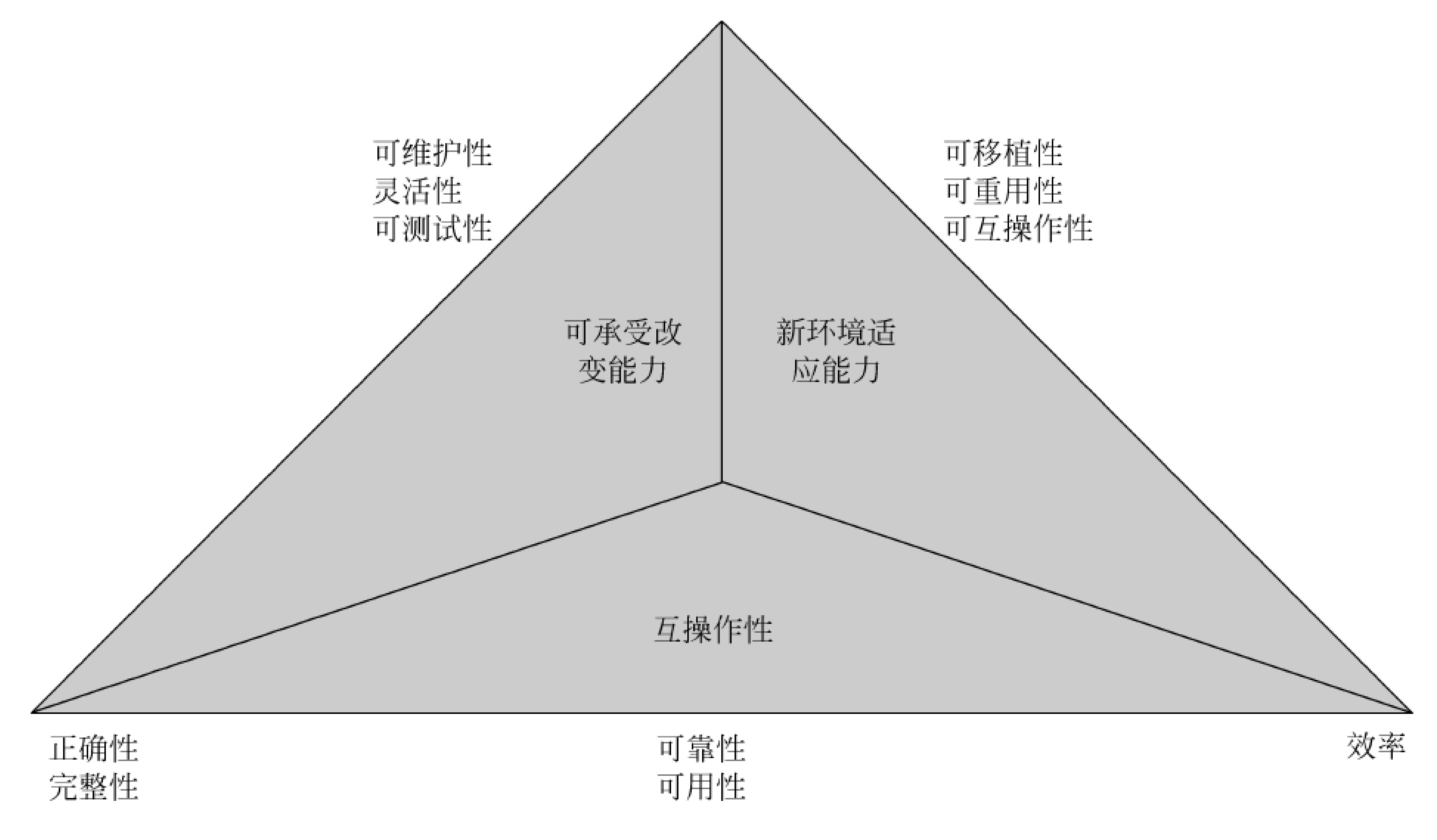


图2-1　McCall模型

J. A. McCall 等人又进一步将软件质量分解到能够度量的程度，提出 FCM 模型，该模型指出影响软件质量的几个最主要的因素，分为软件质量要素(Factor)、衡量标准(Criteria)和量度标准(Metrics)三个层次，这几个因素又是由一些比较低层的如模块化、数据通用性等标准决定的，实际的度量是针对这些标准而言的。该模型描述了因素和它们所依赖的标准之间的一致性。具体内容见表 2-1。他们对软件质量因素进行了研究，认为软件质量是正确性、可靠性、效率等构成的函数，而正确性、可靠性、效率等被称为软件质量因素，或软件质量特征，它表现了系统可见的行为化特征。每一因素又由一些准则来衡量，而准则是跟软件产品和设计相关的质量特征的属性。例如，正确性由可跟踪性、完全性、相容性来判断，每一准则又有一些定量化指标来计量，指标是捕获质量准则属性的度量。Mc-Call 认为软件质量可从两个层次去分析，其上层是外部观察的特性，下层是软件内在的特性。McCall 定义了 11 个软件外部质量特性，称为软件的质量要素，它们是正确性、可靠性、效率、完整性、可使用性、可维护性、可测试性、灵活性、可移植性、重复使用性和连接性。同时，还定义了 22 个软件的内部质量特征，称为软件的质量属性，它们是完备性、一致性、准确性、容错性、简单性、模块性、通用性、可扩充性、工具性、自描述性、执行效率、存储效率、存取控制、存取审查、可操作性、培训性、通信性、软件系统独立性、机器独立性、通信通用性、数据通用性和简明性。软件的内部质量属性通过外部的质量要素反映出来。然而，实践证明以这种方式获得的结果会有一些问题。例如，本质上并不相同的一些问题有可能会被当成同样的问题来对待，导致通过模型获得的反馈也基本相同。这就使得指标的制定及其定量的结果变得难以评价。

表 2-1 McCall 模型内容

层级	名　称	内　容
第一层	质量要素：描述和评价软件质量的一组属性	功能性、可靠性、易用性、效率性、可维护性、可移植性等质量特性，以及将质量特性细化产生的副特性
第二层	衡量标准：衡量标准的组合，反映某一软件质量要素	精确性、稳健性、安全性、通信有效性、处理有效性、设备有效性、可操作性、培训性、完备性、一致性、可追踪性、可见性、硬件系统无关性、软件系统无关性、可扩充性、公用性、模块性、清晰性、自描述性、简单性、结构性、文件完备性等
第三层	量度标准：可由各使用单位自定义	根据软件的需求分析、概要设计、详细设计、编码、测试、确认、维护与使用等阶段，针对每一个阶段制定问卷表，以此实现软件开发过程的质量度量

2．Boehm 模型

1978 年，Boehm 和他的同事们提出了分层结构的软件质量模型，除包含用户的期望和需要的概念，这一点与 McCall 相同之外，还包括 McCall 模型中没有的硬件特性。Boehm 模型始于软件的整体效用，从系统交付后涉及不同类型的用户考虑。第一种用户是初始顾客，系统做了顾客所期望的事，顾客对系统非常满意；第二种用户是要将软件移植到其他软硬件系统下使用的客户；第三种用户是维护系统的程序员。这三种用户都希望系统是可靠有效的。因此，Boehm 模型反映了对软件质量的理解，即软件做了用户要它做的：有效地使用系统资源；易于用户学习和使用；易于测试与维护。

勃姆(Barry W. Boehm)通过将软件质量要素分层定义,在《软件风险管理》(*Software Risk Management*)中首次提出了软件质量度量的层次模型。Boehm 模型将软件的质量特性定义成分层模型,如图 2-2 所示。在表达质量特征的层次性上它与 McCall 模型是非常类似的。不过,它是基于更为广泛的一系列质量特征,它将这些特征最终合并成 19 个标准。Boehm 提出的概念的成功之处在于它包含硬件性能的特征,这在 McCall 模型中是没有的。

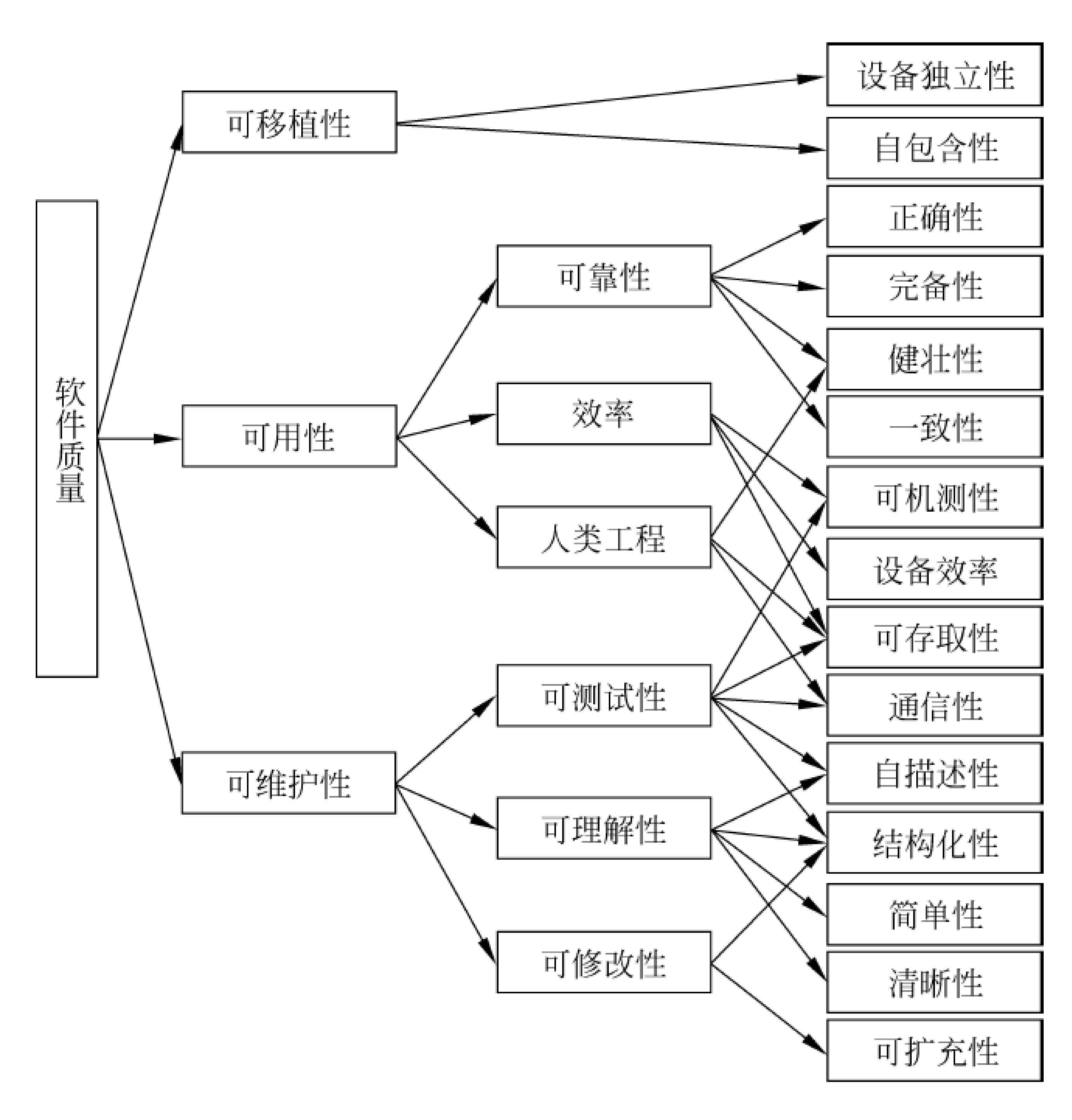


图 2-2　Boehm 模型

3. ISO 9126 模型

20 世纪 90 年代早期,软件工程组织试图将诸多的软件质量模型统一到一个模型中,并把这个模型作为度量软件质量的一个国际标准。国际标准化组织于 1991 年颁布了 ISO 9126—1991 标准《软件产品评价—质量特性及其使用指南》。我国也于 1996 年颁发了同样的软件产品质量评价标准 GB/T 16260—1996,它是一个分层质量模型,有 6 个影响质量的特性。

标准的软件质量测度是这样建立的:软件质量模型分为三个层次,第一层有 6 个影响软件质量的主要因素,在标准中称为“质量特性”。而每个质量特性又可以通过第二层的若干个子特性测量,第二层的每个子特性在评价时要定义并实施若干个度量。当时,ISO 9126 资料性的附录中给出了 21 个子特性。

ISO 9126 的出发点在于使软件最大限度地满足用户的明确的和潜在的需求。这 6 个质量特性最大可能地涵盖了其他早期质量模型中所有的因素,而且彼此交叉最小。软件质量特性与子特性的定义是从用户的角度、开发者的角度和管理者的角度全方位出发考虑的。

因此,ISO 9126—1991 在当时是最为先进、严格的质量模型,它划时代地统一了十几年来国际上推出的各种质量模型。

ISO 9126 质量模型是另一个著名的质量模型,如图 2-3 所示。按照 ISO/IEC 9126—1:2001,该软件质量模型可以分为内部质量和外部质量模型、使用质量模型,而质量模型中又将内部和外部质量分成 6 个质量特性,这 6 个质量特性还可以再继续分成更多的子特征。这些子特征在软件作为计算机系统的一部分时会明显地表现出来,并且会成为内在的软件属性的结果。而另一部分则指定了使用中的质量属性,它们是与针对 6 个软件产品质量属性的用户效果联合在一起的。下面是它给出的软件的 6 个质量特征。

(1) 功能性(Functionality):软件是否满足了客户功能要求。

(2) 可靠性(Reliability):软件是否能够一直在一个稳定的状态上满足可用性。

(3) 可用性(Usability):衡量用户能够使用软件需要多大的努力。

(4) 效率(Efficiency):衡量软件正常运行需要耗费多少物理资源。

(5) 可维护性(Maintainability):衡量对已经完成的软件进行调整需要多大的努力。

(6) 可移植性(Portability):衡量软件是否能够方便地部署到不同的运行环境中。

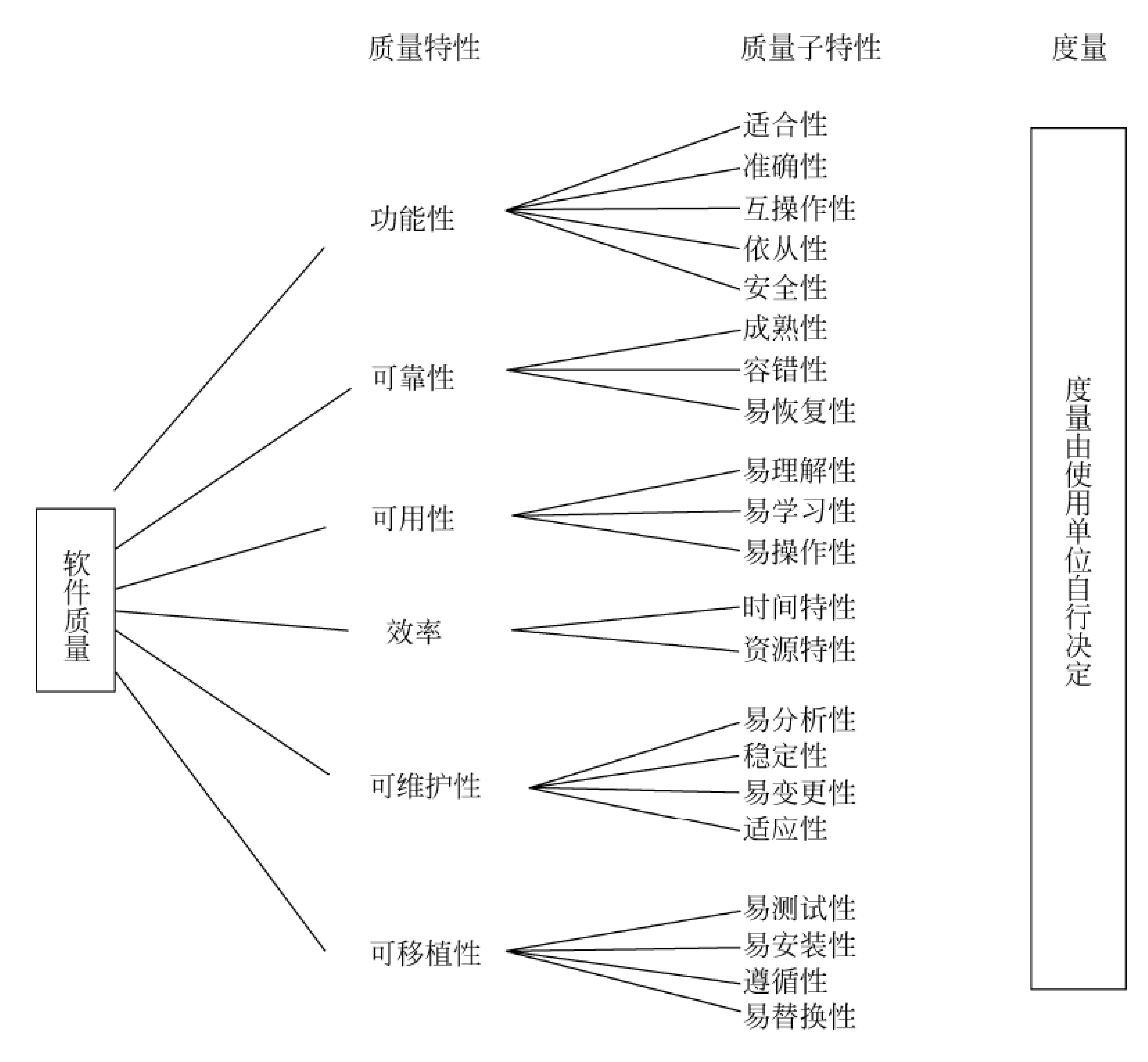


图 2-3 ISO 9126 软件质量度量模型

凯悦(Lawrence E. Hyatt)和罗森贝克(Linda H. Rosenberg)在《识别项目风险以及评价软件质量的软件质量模型与度量》(*A Software Quality Model and Metrics for Identifying Project Risks and Assessing Software Quality*)中比较了这三种最常用的软件质量模型,其基本情况如表 2-2 所示。

表 2-2　软件质量模型比较表

编号	度量标准/目标	麦考尔	勃姆	ISO 9126
1	正确性(Correctness)	×	×	可维护性
2	可靠性(Reliability)	×	×	×
3	完整性(Integrity)	×	×	
4	可用性(Usability)	×	×	×
5	效率性(Efficiency)	×	×	×
6	可维护性(Maintainability)	×	×	×
7	可测试性(Testability)	×		可维护性
8	互操作性(Interoperability)	×		
9	适应性(Flexibility)	×	×	
10	可重用性(Reusability)	×	×	
11	可移植性(Portability)	×	×	×
12	明确性(Clarity)		×	
13	可变更性(Modifiability)		×	可维护性
14	文档化(Documentation)		×	
15	恢复力(Resilience)		×	
16	易懂性(Understandability)		×	
17	有效性(Validity)		×	可维护性
18	功能性(Functionality)			×
19	普遍性(Generality)		×	
20	经济性(Economy)		×	

4. 软件生产能力成熟模型

1993 年，由美国卡内基梅隆大学的软件工程研究所(SEI)创立的 CMM(Capability Maturity Model，软件能力成熟度模型)认证评估，在过去的十几年中，对全球的软件产业产生了非常深远的影响。这是评估软件生产部门(组织、厂家)软件生产能力成熟度的模型，是从软件生产组织过程角度来评估其达到的水平级别。CMM 描述了 5 个级别的软件过程成熟度(初始级、可重复级、已定义级、已管理级、优化级)，分别标志着软件企业能力成熟度的 5 个层次，如图 2-4 所示。成熟度反映了软件过程能力(Software Process Capability)的大小，任何一个软件机构的软件过程必定属于其中某个级别。

1）初始级

初始级(Initial)的软件过程是未加定义的随意过程，项目的执行是随意甚至是混乱的。也许，有些企业制定了一些软件工程规范，但若这些规范未能覆盖基本的关键过程要求，且执行没有政策、资源等方面的保证时，那么它仍然被视为初始级。

初始级的特点是：工作无序，项目进行过程中常放弃开始制订的计划；管理无章，缺乏健全的管理制度；开发项目成效不稳定，产品的质量和性能严重依赖于个人的能力和行为。

2）可重复级

根据多年的经验和教训，人们总结出软件开发的首要问题不是技术问题而是管理问题。因此，第二级的焦点集中在软件管理过程上。一个可管理的过程则是一个可重复的过程，可重复的过程才能逐渐改进和成熟。可重复级(Repeatable)的管理过程包括需求管理、项目

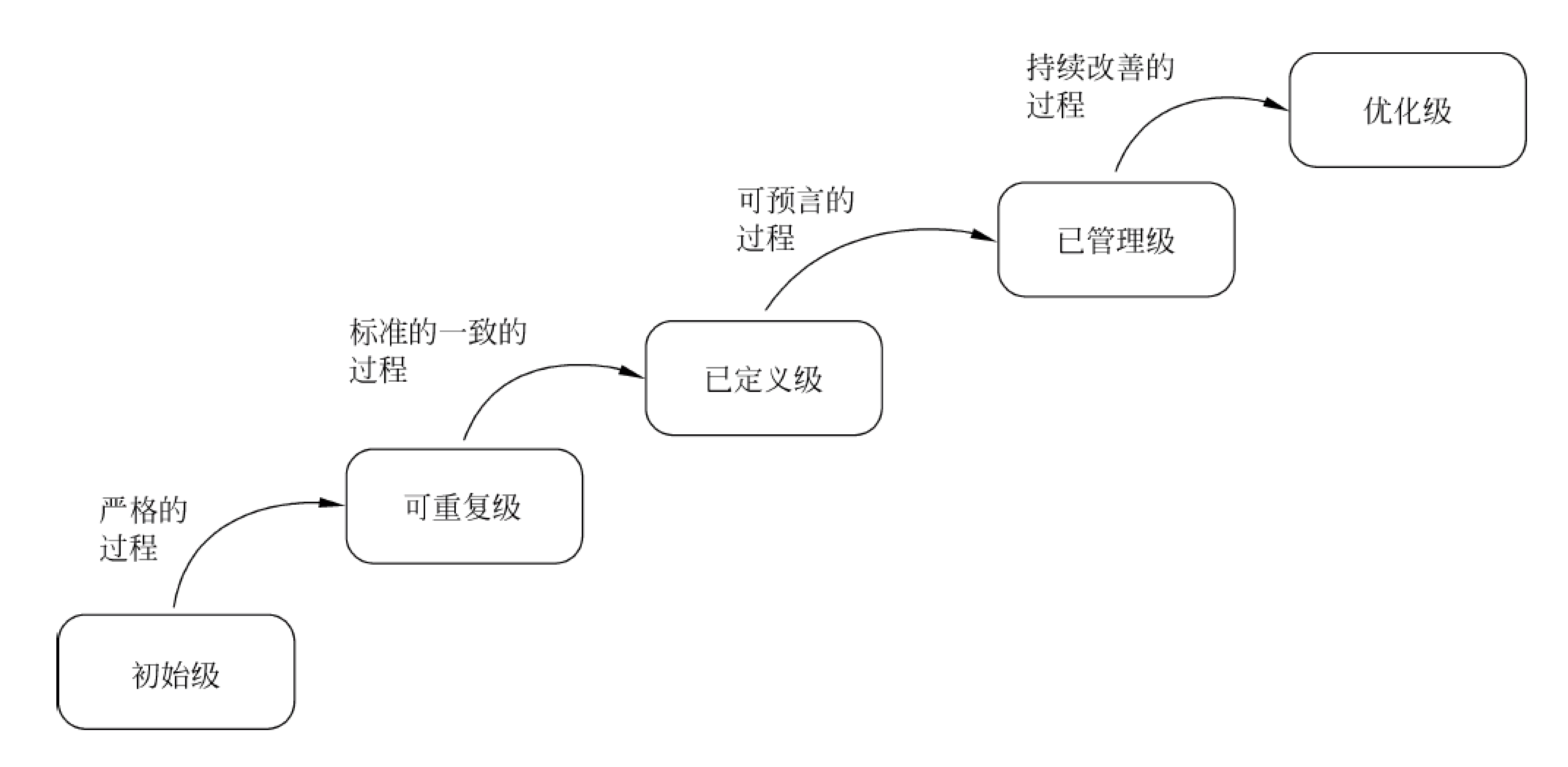


图 2-4 软件能力成熟度模型

管理、质量管理、配置管理和子合同管理 5 个方面；其中，项目管理过程又分为计划过程和跟踪与监控过程。通过实施这些过程，从管理角度可以看到一个按计划执行的且阶段可控的软件开发过程。

可重复级的特点是：管理制度化，建立了基本的管理制度和规程，管理工作有章可循；初步实现标准化，开发工作较好地实施标准；变更均依法进行，做到基线化；稳定可跟踪，新项目的计划和管理基于过去的经验，具有重复以前成功项目的环境和条件。

3）已定义级

在可重复级定义了管理的基本过程，而没有定义执行的步骤标准。在第三级则要求制定企业范围的工程化标准，并将这些标准集成到企业软件开发标准过程中。所有开发的项目需根据这个标准过程，裁剪出与项目适宜的过程，并且按照过程执行。过程的裁剪不是随意的，在使用前必须经过企业有关人员的批准。

已定义级(Defined)的特点是：开发过程，包括技术工作和管理工作，均已实现标准化、文档化；建立了完善的培训制度和专家评审制度；全部技术活动和管理活动均稳定实施；项目的质量、进度和费用均可控制；对项目进行中的过程、岗位和职责均有共同的理解。

4）已管理级

第四级的管理是量化的管理。所有过程需建立相应的度量方式，所有产品的质量(包括工作产品和提交给用户的最终产品)需要有明确的度量指标。这些度量应是详尽的，且可用于理解和控制软件过程和产品。量化控制将使软件开发真正成为一种工业生产活动。

已管理级(Managed)的特点是：产品和过程已建立了定量的质量目标；过程中活动的生产率和质量是可度量的；已建立过程数据库；已实现项目产品和过程的控制；可预测过程和产品质量趋势，如预测偏差，实现及时纠正。

5）优化级

优化级(Optimizing)的目标是达到一个持续改善的境界。所谓持续改善是指可以根据过程执行的反馈信息来改善下一步的执行过程，即优化执行步骤。如果企业达到了第 5 级，就表明该企业能够根据实际的项目性质、技术等因素，不断调整软件生产过程以求达到

最佳。

优化级的特点是：可集中精力改进过程，采用新技术、新方法；拥有防止出现缺陷、识别薄弱环节以及加以改进的手段；可取得过程有效性的统计数据，并可据此进行分析，从而得出最佳方法。

CMM 的每个等级是通过三个层次加以定义的：关键过程域、关键实践类、关键实践。

在框架的某一“平台”上，其实施将对达到下一成熟度等级的目标起保证作用的过程域，被称为关键过程域。

关键实践描述了对关键过程域的实施起关键作用的方针规程、措施、活动以及相关的基础设施。关键实践的作用是一个关键过程域中所包含的关键实践的实施，才可以达到该关键过程域中的目标。

关键过程域的目标表明了一个关键过程域的范围、边界和意图。例如，软件项目规划是 2 级的一个关键过程域。其目标可概括为：制定进行软件工程和管理软件项目的合理计划。这些计划是管理软件项目的必要基础，促进按选定的软件生存周期模型分阶段分工地进行软件开发。按阶段组织检查，实施控制。具体包括：软件生存周期已选定，并经评审确认；对计划中的软件规模、工作量、成本、风险等已经进行估计；软件项目的活动和约定是有计划的；影响计划进度的关键路径是已标识的且受控的；影响计划进度的关键资源需求是已标识的；在软件生存周期的里程碑处，对计划的执行有检查、有记录，问题有报告；对于介入软件开发计划的软件负责人、软件工程师和有关人员进行了软件估计和计划方面的培训；选定有可管理规模的、预定阶段的软件生存周期模型，如瀑布型、增量型、渐进型、螺旋型、逆向工程型；标识为控制软件项目所必需的软件工作产品。

每一级的关键实践按“共同特征”予以组织。这些“共同特征”为：实施承诺——制定方针政策；实施能力——确保必备条件；实施活动——实施软件过程；度量分析与验证——检查实施情况。也就是说，每一关键过程域包含一组关键实践，并按“共同特征”组织为每一级的关键实践类——制定方针政策，确保必备条件，实施软件过程，检查实施情况。

关键过程域、关键实践类、关键实践之间的关系如图 2-5 所示，就是：实施软件过程这一关键实践类中的关键实践，描述了为建立过程能力，过程实施者必须做些什么；其他关键实践类中的实践，作为一个整体使实施软件过程中的实践规范化。

除去初始级以外，其他 4 级都有若干个引导软件机构改进软件过程的要点，称为关键过程域（Key Process Area，KPA）。每一个关键过程域是一组相关的活动，成功地完成这些活动，将会对提高过程能力起重要作用。图 2-6 给出了各成熟度级别对应的关键过程域。各级包含的关键过程域如下。

（1）可重复级：6 个。包含软件配置管理，软件质量保证、子合同管理、项目跟踪和监督、软件项目规划、需求管理。

（2）已定义级：6＋7(个)＝13 个。除了可重复级的关键过程域外，还包括同行评审、组间协调、软件产品工程、集成软件管理、培训管理、机构过程定义、机构过程焦点。

（3）已管理级：13＋2(个)＝15 个。除了已定义级的关键过程域外，还包括软件质量管理、过程的量化管理。

（4）持续优化级：15＋3(个)＝18 个。除了已管理级的关键过程域外，还包括过程变更管理、技术变更管理、缺陷预防。

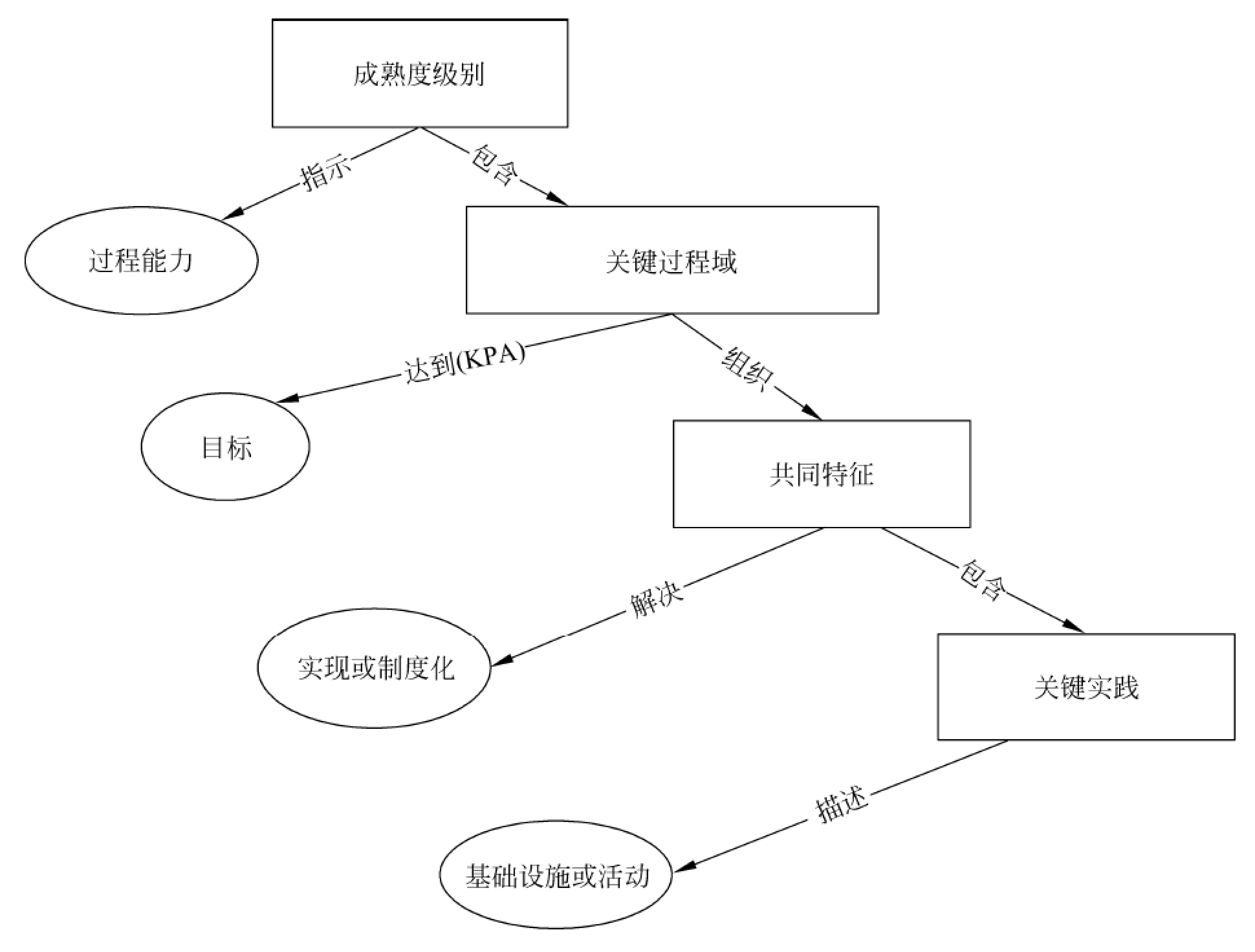


图 2-5 关键过程域、关键实践类、关键实践之间的基本关系

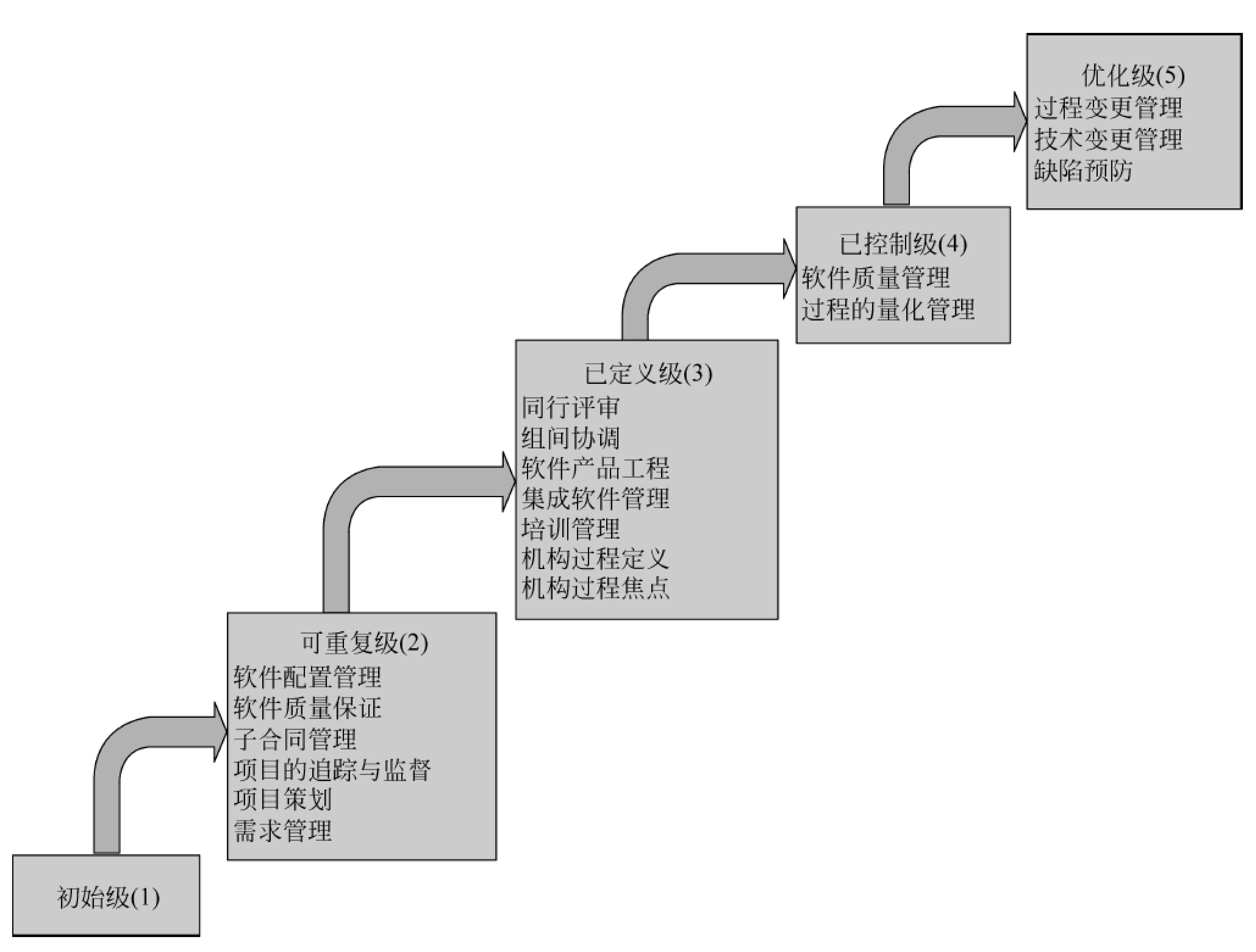


图 2-6 各级包含的关键过程域

从低到高，软件开发生产计划精度逐级升高，单位工程生产周期逐级缩短，单位工程成本逐级降低。据SEI统计，通过评估的软件公司对项目的估计与控制能力约提升40%～50%；生产率提高10%～20%，软件产品出错率下降超过1/3。

一般来说，通过CMM认证的级别越高，其越容易获得用户的信任，在国内、国际市场上的竞争力也就越强。因此，是否能够通过CMM认证也成为国际上衡量软件企业工程开发能力的一个重要标志。

CMM是目前世界公认的软件产品进入国际市场的通行证，它不仅是对产品质量的认证，更是一种软件过程改善的途径。该模型被认为是第一个集成化的模型。

为了以示区别，国内外很多资料把CMM叫作SW-CMM。目前，SEI研制和保有的能力成熟度模型有以下几种。

(1) 软件集成能力成熟度模型(Capability Maturity Model Integration(Service Mark), CMMI)；

(2) 软件能力成熟度模型(Capability Maturity Model(r) for Software, SW-CMM)；

(3) 人力能力成熟度模型(People Capability Maturity Model, P-CMM)；

(4) 软件采办能力成熟度模型(Software Acquisition Capability Maturity Model, SA-CMM)；

(5) 系统工程能力成熟度模型(Systems Engineering Capability Maturity Model, SE-CMM)；

(6) 一体化生产研制能力成熟度模型(Integrated Product Development Capability Maturity Model, IPD-CMM)。

建立这些模型的指导思想和方法论都是一样的，即评估能力、发现问题、帮助改进。

现有的这些质量模型总是意图能够适用于所有类型的软件开发，成为一个通用的模型。然而，软件质量是非常复杂的，很难定义出一个能够适用于所有软件质量度量的模型。每个软件系统都有它自己的特征，因此在使用质量模型时必须考虑各类应用的特殊需求。并且，由于计算机应用的飞速发展，人们需要寻找不仅能够在软件质量管理方面提供有效帮助，而且还能够对软件开发中的其他活动提供相应支持的质量模型。

总之，以上这些通用的软件质量模型定义的标准被应用于各种软件，包括计算机程序和包含在固件中的数据。那些特征和子特征为软件产品质量提供了一致的术语，并且为制定软件的质量需求和确定软件性能的平衡点指定了一个框架。然而，层次模型由一些质量属性、标准及准则等构成，它们只表达了质量属性之间一些正面的影响关系，对于那些更复杂的关系它们却无能为力。关系模型能够表达质量属性之间正面、反面及中立的影响，但对于一些更为复杂的关系则同样无法表达。并且，它们还有一个相同的弱点，就是现有的这些质量模型总是意图能够适用于所有类型的软件开发，成为一个通用的模型。然而，正像上文提到的那样，软件质量是非常复杂的，很难定义出一个能够适用于所有软件质量度量的模型。每个软件系统都有它自己的特征，在使用质量模型时必须考虑各类应用的特殊需求。并且，由于计算机应用的飞速发展，人们需要寻找不仅能够在软件质量管理方面提供有效帮助，而且还能够对软件开发中的其他活动提供相应支持的质量模型。

2.4 标准的发展

随着信息技术应用的不断增长，关键的计算机系统的数量也在增长。这些系统包括安全、生活、经济以及保密方面的关键系统。这些系统的软件质量尤其重要，因为软件的故障可能导致非常严重的后果。

纵观整个软件工程的历史，提高软件质量已成为最重要的目标。评价软件产品的质量对获取和开发满足质量需求的软件是不可缺少的，各种软件质量特性的相关重要性取决于作为整体一部分的系统的任务和目标，需要评价软件产品以判断其相关的质量特性是否满足系统的需求。

到1999年，国际软件工程标准化组织将软件"产品评价"与"产品质量"分为两个标准："产品评价"注重软件质量评价的支持和评价过程；"产品质量"注重软件本身的质量度量模型。

软件质量评价的基本部分包括质量模型、评价方法、软件的测量和支持工具。要想开发好的软件，就要规定质量需求，策划、实现和控制软件质量保证过程，应评价中间产品和最终产品。要达到评价软件质量的目的，应用有效的度量方法测量软件的质量属性是非常必要的。

GB/T 18905—2002(ISO 14598—1999)《软件工程产品评价》系列标准为软件产品质量的测量、评估和评价提供了方法。它所描述的既不是软件生产过程的评价方法，也不是预算成本的方法(软件产品的质量测量当然可以用于这两个目的)。

因为质量特性和相关的度量不仅可用于软件产品评价，而且也可用于质量需求的定义以及其他用途，国际软件工程标准化组织颁布了ISO 14598—1999《软件工程产品评价》，而在2001年修订了ISO 9126—1991标准。

所以，早期的GB/T 16260—1906(ISO 9126—1991)《软件产品评价—质量特性及其使用指南》已经被两个相关的由多部分组成的标准：GB/T 18905—2002(ISO 14598—1999)《软件工程产品评价》和GB/T 16260—2003 (ISO 9126—2001)《软件工程产品质量》所取代。

1. GB/T 18905 产品评价

1) GB/T 18905 基本组成

GB/T 18905—2002《软件工程产品评价》，该系列标准由以下6部分组成。

GB/T 18905.1 《软件工程 产品评价》第1部分 概述

GB/T 18905.2 《软件工程 产品评价》第2部分 策划和管理

GB/T 18905.3 《软件工程 产品评价》第3部分 开发者用的过程

GB/T 18905.4 《软件工程 产品评价》第4部分 需方用的过程

GB/T 18905.5 《软件工程 产品评价》第5部分 评价者用的过程

GB/T 18905.6 《软件工程 产品评价》第6部分 评价模块的文档编制

2) 评价者用的过程(GB/T 18905.5)

(1) 开发者用的过程

计划开发新产品或增强现有的产品，以及打算利用他们自己的技术人员进行产品评价的组织应使用GB/T 18905.3。这部分主要强调使用那些能预测最终产品质量的指标，这些指标将通过虚盘在生存期期间开发的中间产品得到。

（2）需方用的过程

计划获取或受用某个已有的软件产品或预先开发的软件产品的组织，应使用GB/T 18905.4。该部分可用来决定接受产品或者从众多可选产品中选择某个产品（产品可以是自包含的，或是系统的一部分，或者是较大产品的一部分）。

（3）评价者用的过程

对软件产品执行独立评估的评价者应使用GB/T 18905.5。这种评价是应开发者、需方或其他方的请求来进行的。这部分将由那些执行独立评价的人员使用，他们通常为第三方组织工作。

3）关于评价支持

（1）策划与管理

GB/T 18905.2为策划和管理部分，包含对软件产品评价的支持功能的需求和指南。这些支持与策划和管理软件评价过程及相关的活动有关，包括组织内评价专业知识的开发、获取、标准化、控制、转换和反馈。

（2）评价模块

GB/T 18905.6为编制评价模块的文档提供指南。这些模块包括质量模型的规范（即特性、子特性和相应的内部或外部度量），与模型计划的应用有关的数据、信息和与模型的实际应用有关的信息。

4）通用评价过程

软件产品的一般评价过程是：确立评价需求，然后规定、设计与执行评价，如图2-7所示。

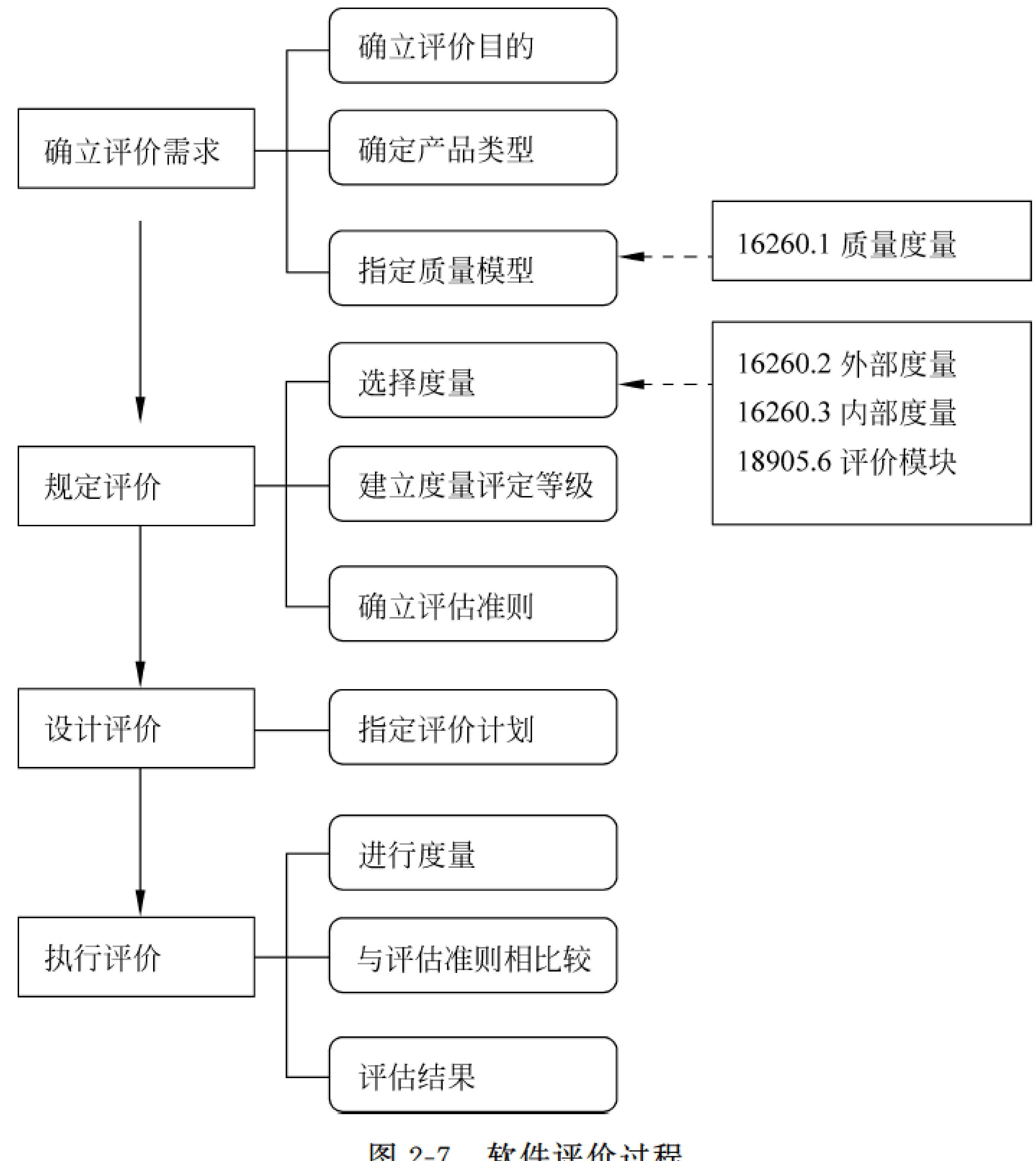


图2-7　软件评价过程

5）评价需求

软件质量评价的目的是为了直接支持开发和获得能满足用户和消费者要求的软件。最终目标是保证产品能提供所要求的质量，即满足用户（包括操作者、软件结果的接受者或软件的维护者）明确和隐含的要求。

(1) 评价中间产品质量的目的

① 决定（是否）接受分包商交付的中间产品；

② 决定某个过程的完成，以及何时把产品送交下一个过程；

③ 预计或估计最终产品的质量；

④ 收集中间产品的信息以便控制和管理过程。

(2) 评价最终产品质量的目的

① 决定（是否）接受产品；

② 决定何时发布产品；

③ 与相互竞争的产品进行比较；

④ 从众多可选的产品中选择一种产品；

⑤ 使用产品时评估产品积极和消极的影响；

⑥ 决定何时增强或替换该产品。

6）确定要评价产品的类型

要评价的中间或最终软件产品的类型取决于所处的生存周期的阶段和评价的目的，如图 2-8 所示。

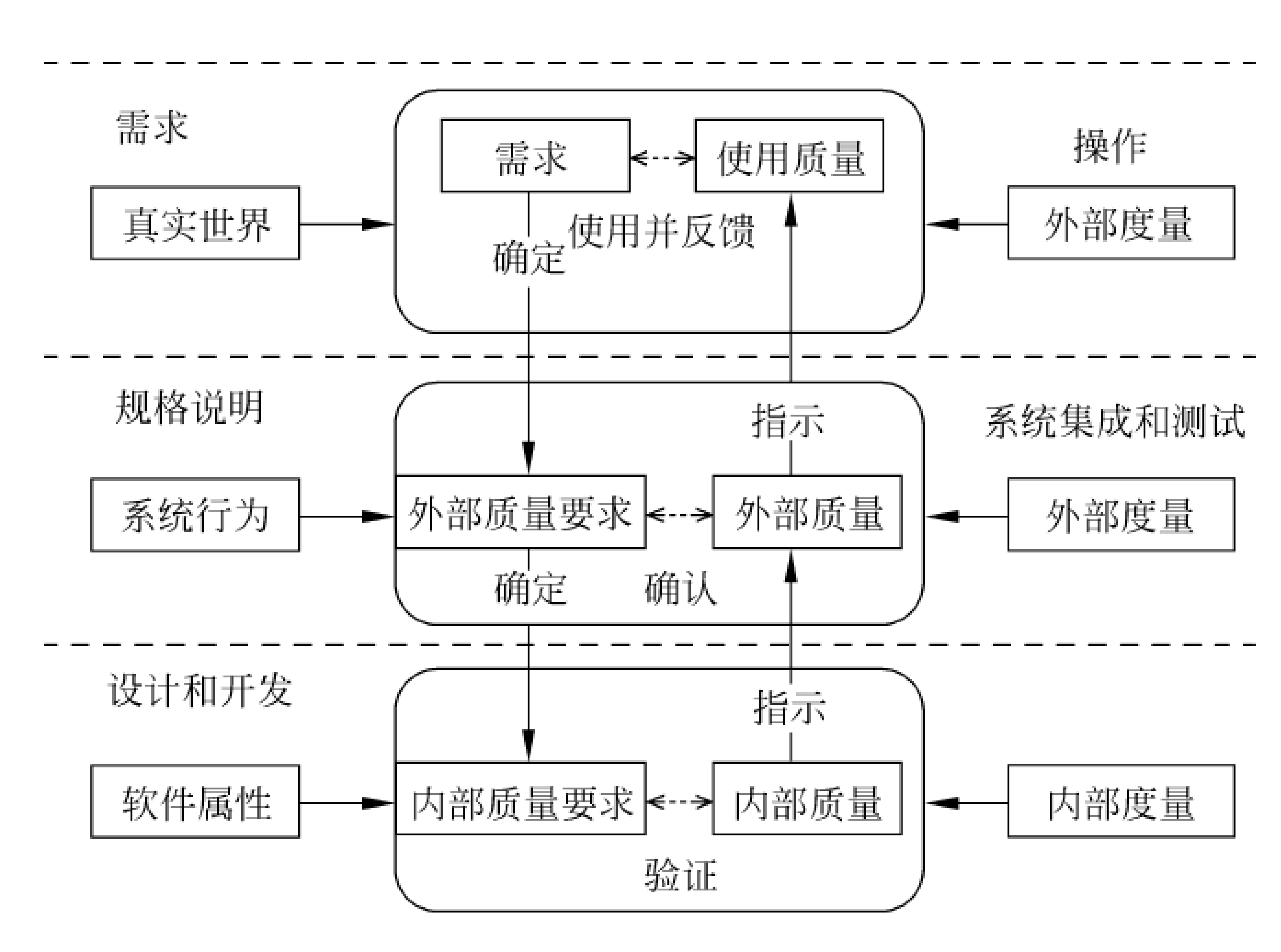


图 2-8 软件生存周期中的质量

7）质量之间的关系

质量之间的关系如图 2-9 所示。

将软件的内部质量属性与外部质量需求联系起来是十分重要的，使得开发中的软件产品（中间和最终的软件产品）的质量特性可以根据最终系统使用质量需求来进行评估。内部质量的值很低，除非有证据表明它与外部质量有关。

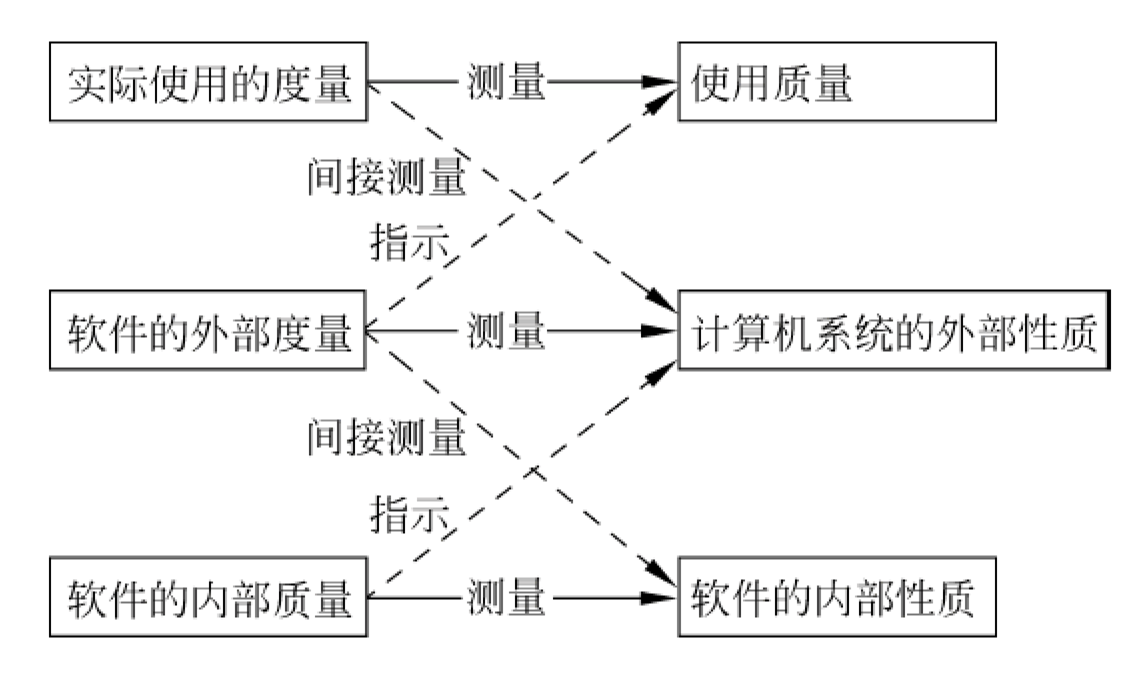


图 2-9　质量之间的关系

8）规定质量模型

软件评价所用的质量模型通常代表软件质量属性的总体，这些质量属性用特性和子特性的分层树结构进行分类。该结构的最高级由质量特性构成，最低级由软件质量属性构成。GB/T 16260.1 提供一个通用模型，它定义了 6 种软件质量特性，包括功能性、可靠性、易用性、效率、可维护性和可移植性，如图 2-10 所示。

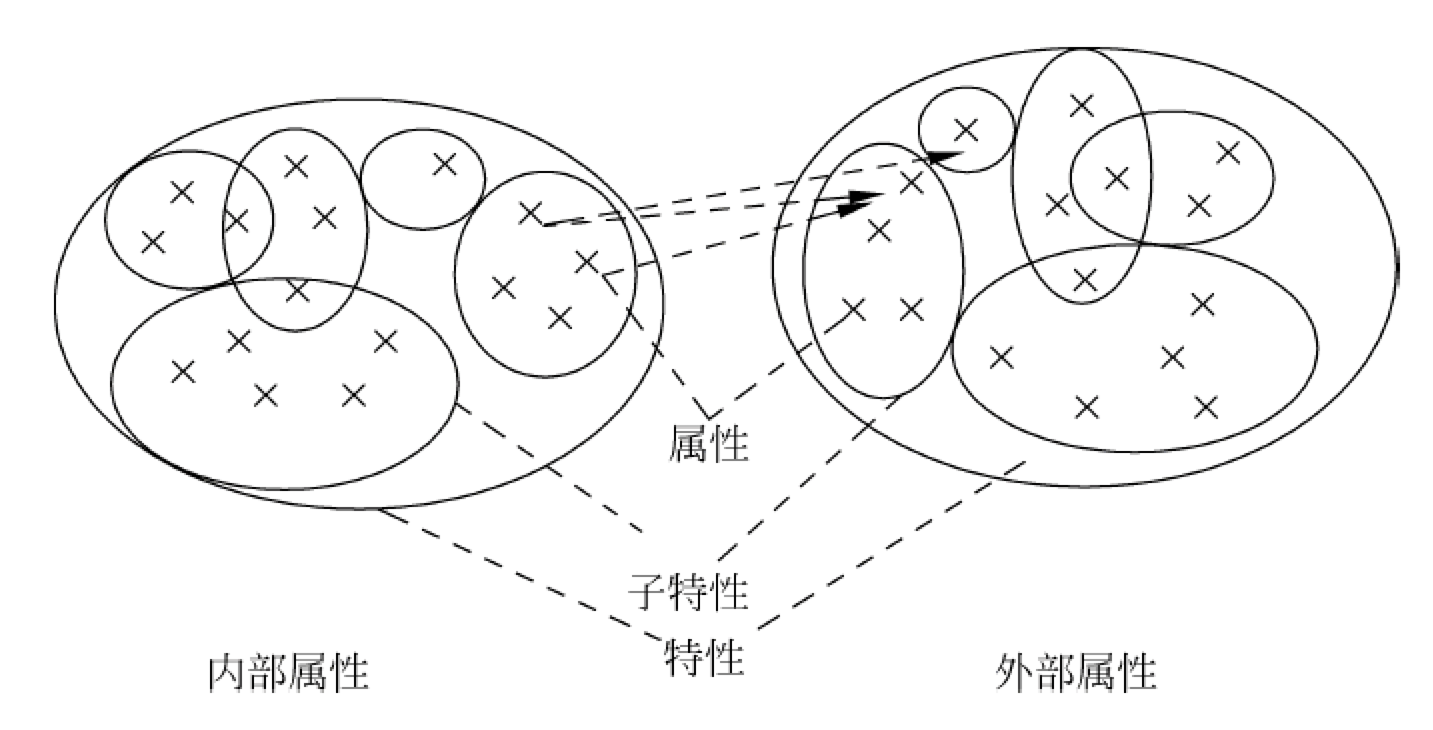


图 2-10　质量特性、子特性和属性

软件产品的内部质量属性是软件产品的可测量的性质，这些性质影响产品满足明确和隐含要求的能力，可以用一个或更多的属性来评估一个特定的软件质量特性或子特性。

9）规定评价

(1) 选择度量

质量特性的定义方式不允许对它们进行直接测量。需要建立与软件产品特性相关的度量。与某个质量特性相关的每个可量化的软件内部属性和每个可量化的软件外部属性，与其软件环境进行相互作用，能被确立为一种度量。

质量可以随环境和应用度量的开发过程阶段的不同而有所区别，用于开发过程的度量应与用户观点的度量有关，因为从用户视角出发的度量是至关重要的。

(2) 测量的种类

评价有两个主要目的：确定问题以便解决问题；与可替代的产品进行比较，或对照需求比较产品质量。

(3)确定度量评价等级

可量化的特征可以用度量质量的方法进行定量的测量。其结果是，将测量值映射到某

一标度上。这个值本身并不表示满意的等级，因此，这一标度必须根据需求的不同满意度级别分为不同的范围。例如，将标度分成两类：满意和不满意。将标度分成4类：针对已有产品或可替换产品的当前级、最差缓和计划级划分成4类，即超出要求，目标范围、可接受的最低限度、不可接受。定义当前级是为控制新系统不因当前状况而恶化；计划级是指一旦资源可利用，产品即可获得；最差级是指万一产品不符合计划级时用户的可接受边界，如图2-11所示。

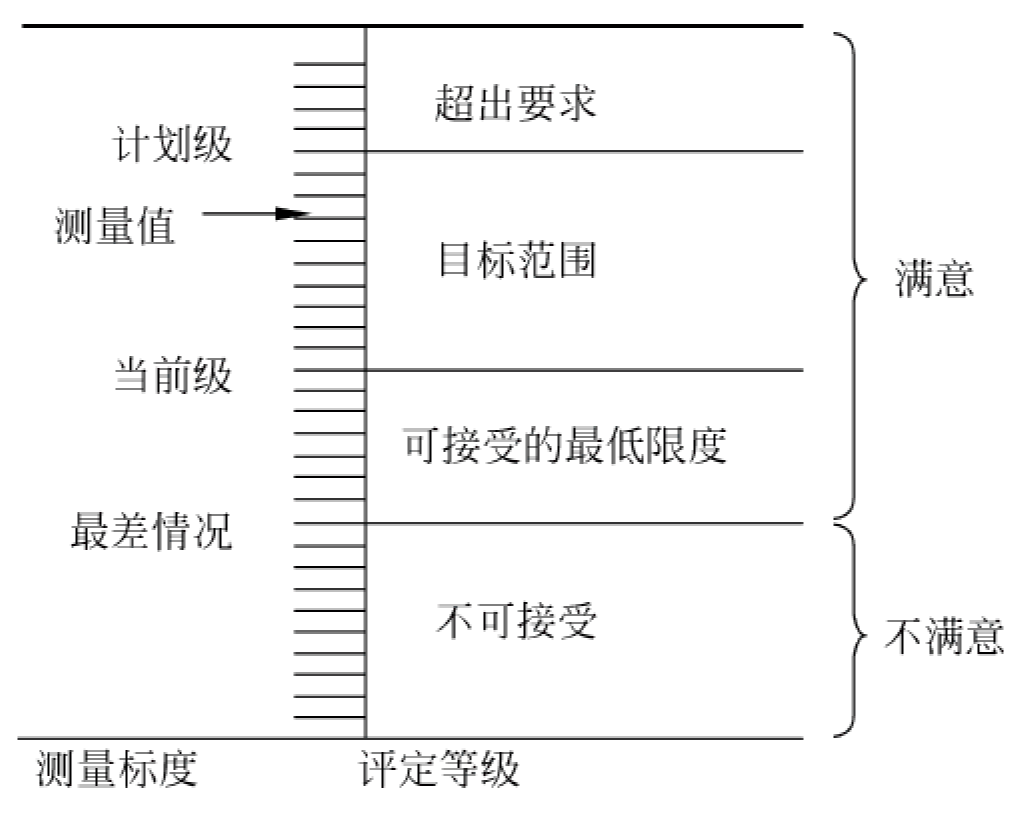


图2-11　度量的等级

(4) 确定评估准则

软件质量需求规格说明应使用定义良好的、适当的质量模型来表示。为此，除非有特殊原因需使用其他模型外，应使用GB/T 16260.1中的质量模型和定义。

为了评估产品质量，需要总结针对不同特性的评价结果。评价者应为此准备一个规程，其中，对不同的质量特性使用不同的评价准则，每个质量特性又以数个子特性或子特性的加权组合来说明，规程通常还包括时间和成本等有助于在特定环境下评估软件产品质量的其他方面。

2. GB/T 16201.1 产品质量

1) 基本组成

GB/T 16260—2003《软件工程产品质量》，该系列标准由以下4部分组成。

GB/T 16260.1　《软件工程　产品质量》第1部分 质量模型

GB/T 16260.2　《软件工程　产品质量》第2部分 外部质量

GB/T 16260.3　《软件工程　产品质量》第3部分 内部质量

GB/T 16260.4　《软件工程　产品质量》第4部分 使用质量度量

2) 标准概述

(1) 标准的变化

国际上对ISO 9126进行了修订，保留了原来的6个软件质量特性，定义了一个通用的质量模型，并给出了度量的例子，与原标准相比，其主要区别在于：

① 质量特性中增加了使用质量特性；

② 质量度量分为外部度量、内部度量和使用质量度量；

③ 子特性作为标准的一部分，在原版标准资料性附录中的子特性基础上增加了一些；

④ 删除了评价过程内容(已在 ISO 14598 中进行了说明)；

⑤ 与 ISO 14598 的内容基本协调。

(2) 标准之间的关系

GB/T 18905.1 概述是产品评价标准的总则，GB/T 16260 的评价过程与度量是遵循 GB/T 18905 的。GB/T 18905 和 GB/T 16260 系列标准之间的关系如图 2-12 所示。

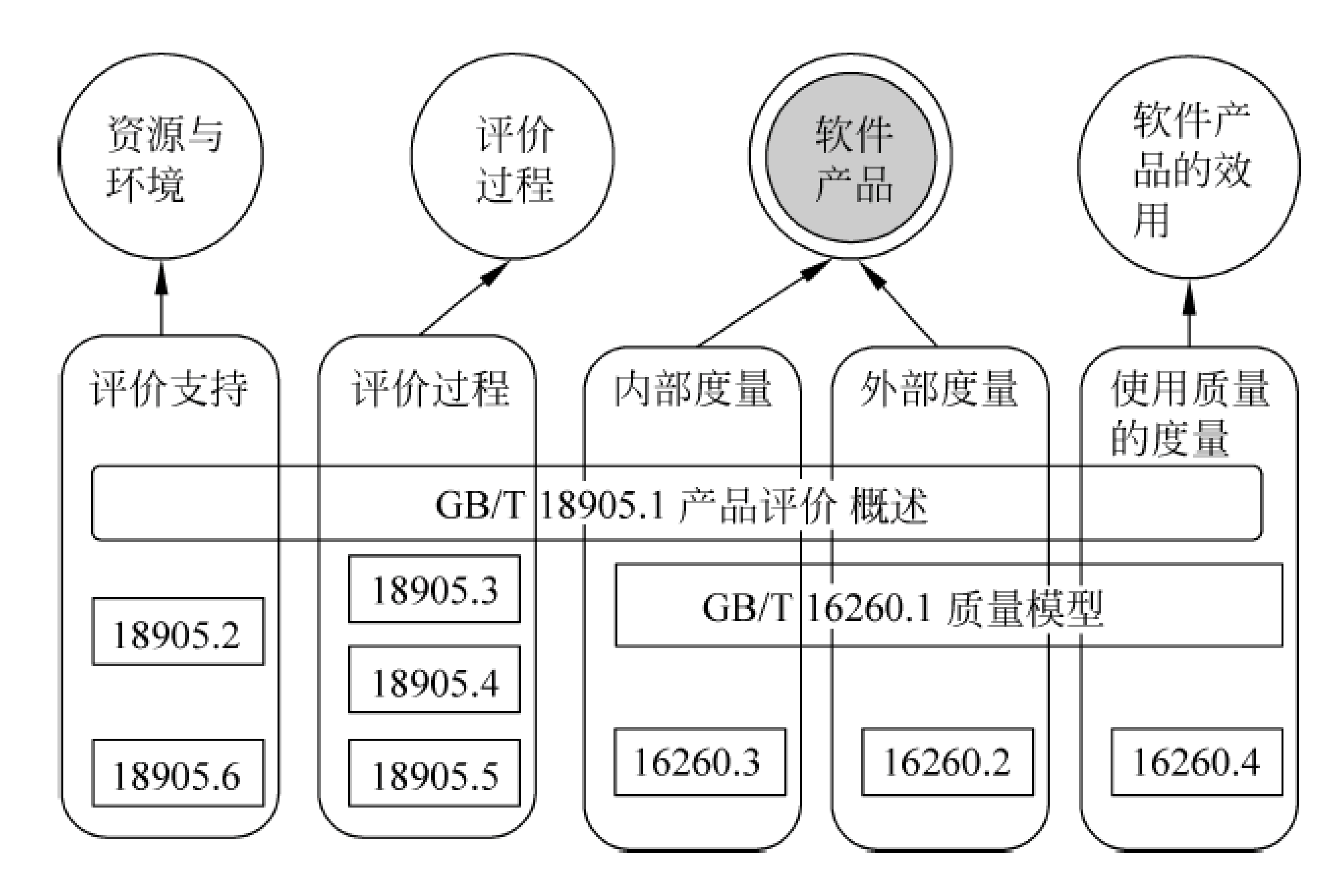


图 2-12 GB/T 18905 和 GB/T 16260 系列标准之间的关系

3) 标准的范围

标准可从软件的获取、需求、开发、使用、评价、支持、维护、质量保证和审核相关的不同观点来确定和评价软件产品的质量。可以被开发者、需方、质量保证人员和独立评价者，特别是那些对确定和评价软件产品质量负责的人员所使用。

本标准中定义的质量模型使用实例是：

(1) 确定需求定义的完整性；

(2) 确定软件需求；

(3) 确定软件设计目标；

(4) 确定软件测试目标；

(5) 确定质量保证准则；

(6) 为完整的软件产品确定验收准则。

软件质量标准还可以和软件质量保证过程，以及软件过程改进有关的标准一起使用。

4) 质量模型框架

(1) 软件质量特性与度量

① 质量特性和子特性；

② 外部度量；

③ 内部度量。

GB/T 16260.1 定义了质量特性、相关的子特性以及 GB/T 16260 质量模型上面三层之间的关系。GB/T 16260.2 和 GB/T 16260.3 确定了每种度量(外部和内部的)与其相应的

特性和子特性之间的关系。注意，某些内部度量有对应的外部度量。软件质量特性与度量如图 2-13 所示。

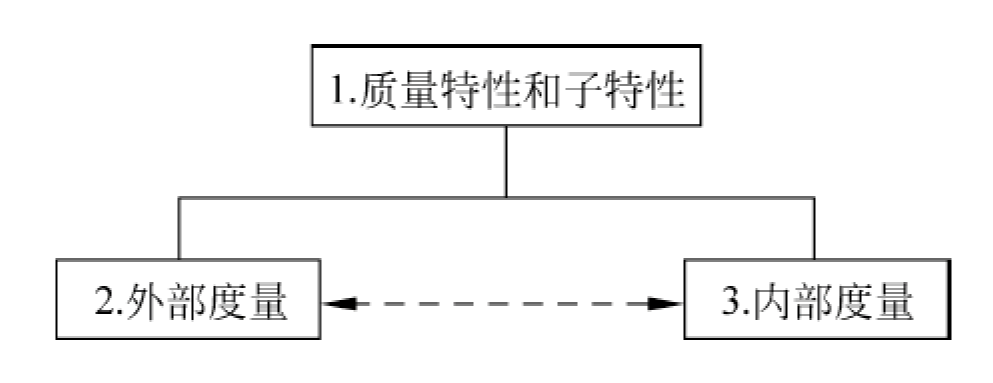


图 2-13 软件质量特性与度量

(2) 质量途径

新版 GB/T 16260 的用户质量要求包括：在特定的使用环境和条件下对使用质量的需求，而在规定外部质量和内部质量的特性和子特性时也可以使用这些需求。

软件产品质量可以通过测量内部属性，或者测量外部属性，或者测量使用质量的属性来评价，目标就是使产品在特定的使用环境和条件下具有所需的效用。生存周期的质量途径如图 2-14 所示。

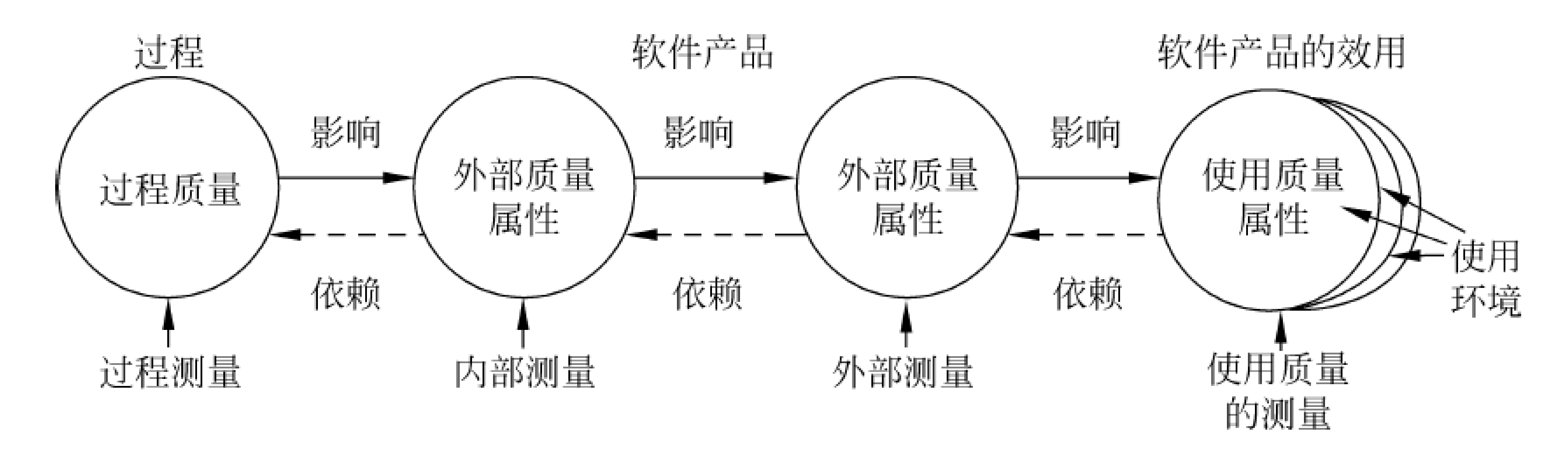


图 2-14 生存周期的质量途径

过程质量有助于提高产品质量，而提高产品质量有助于提高使用质量，因此，评估和改善过程是提高产品质量的一种手段，而评价和改进产品质量是提高使用质量的一种方法。同样，评价使用质量可以为改进产品提供反馈，而评价产品则可以为改善过程提供反馈。适当的软件内部属性是获得所需外部特性的先决条件；而适当的外部特性是获得使用质量的先决条件。

软件产品质量需求包括内部质量、外部质量和使用质量的评估准则，以满足开发者、维护者、需求方以及最终用户的需求。

(3) 产品质量和生存周期

内部质量、外部质量和使用质量的观点在软件生存周期中是变化的。

在软件生存周期的不同阶段存在着关于产品质量和相关度量的不同观点。软件生存周期的质量如图 2-15 所示。

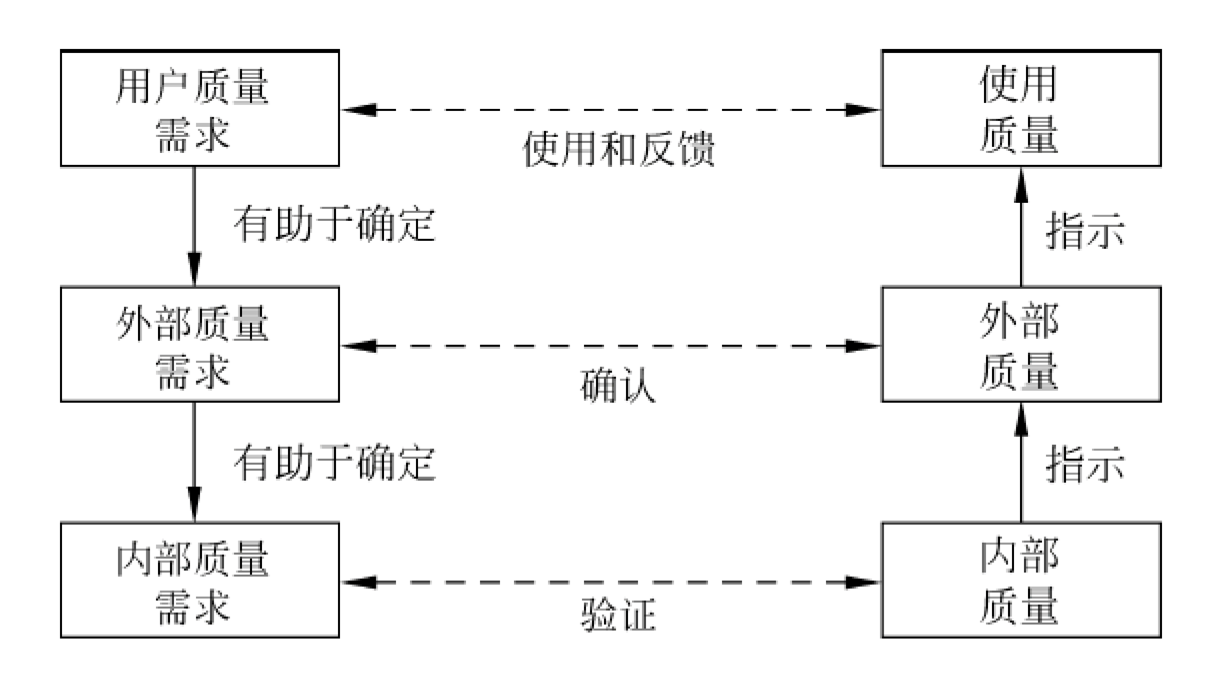


图 2-15 软件生存周期的质量

① 用户的质量要求

可按使用质量的度量、外部度量或内部度量来规定质量需求。当验证产品时，这些由质量规定的需求应作为准则使用。

② 外部质量要求

从外部观点来规定必需的质量级别，包括来源于用户质量要求（使用质量要求）。外部质量需求用作不同开发阶段的验证目标。

③ 内部质量要求

内部质量需求是从产品的内部观点来规定必需的质量水平。内部质量需求用来规定中间产品的属性，包括静态的和动态的模型、其他的文档和源代码。内部质量需求可用作不同开发阶段的验证目标。

④ 使用质量

使用质量是从用户观点出发，来看待软件产品用于特定环境和条件下的质量，它测量用户在特定环境中达到其任务目标的程度，而不是测量软件自身的性质。

⑤ 外部质量

外部质量是从外部观点出发的软件产品特性的总体。它是当软件执行时，更典型地使用外部度量在模拟环境中，用模拟数据测试时所被测量和评价的质量。

⑥ 内部质量

内部质量是从内部观点出发的软件产品特性的总体，内部质量是针对内部质量需求被测量和评价的质量。

5）外部质量和内部质量的质量模型

外部质量和内部质量的质量模型软件，其质量属性划分为 6 种特性（功能性、可靠性、易用性、效率、维护性和可移植性），并进一步细分为一些子特性。外部质量和内部质量的质量模型如图 2-16 所示。

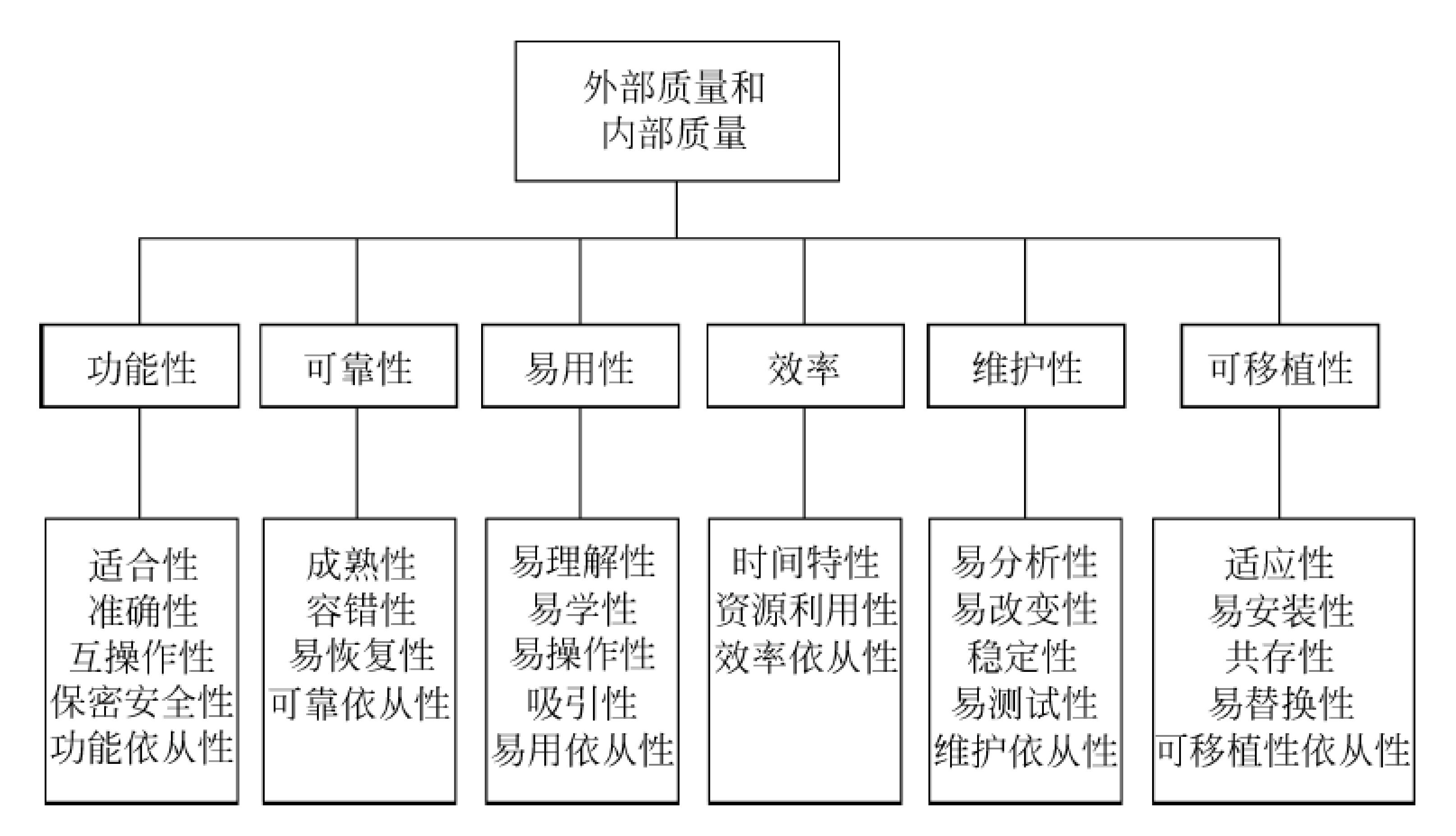


图 2-16　外部质量和内部质量的质量模型

（1）功能性

功能性是指当软件在指定条件下使用时，软件产品满足明确和隐含要求功能的能力。

① 适合性

适合性指软件产品为指定的任务和用户目标提供一组合适的功能的能力。

② 准确性

准确性指软件产品提供具有所需精度的正确或相符的结果或效果的能力。

③ 互操作性

互操作性指软件产品与一个或更多的规定系统进行交互的能力。

④ 保密安全性

保密安全性指软件产品保护信息和数据的能力,以使未授权的人员或系统不能阅读或修改这些信息和数据,但不拒绝授权人员或系统对它们的访问。

⑤ 功能依从性

功能依从性指软件产品依附于同功能性相关的标准、约定或法规以及类似规定的能力。

(2) 可靠性

① 成熟性

成熟性指软件产品避免因软件中错误的发生而导致失效的能力。

② 容错性

容错性指软件发生故障或者违反指定接口的情况下,软件产品维持规定的性能级别的能力。

③ 易恢复性

易恢复性指在失效发生的情况下,软件产品重建规定的性能级别并恢复受直接影响的数据的能力。

④ 可靠依从性

可靠依从性指软件产品依附于同可靠性相关的标准、约定或规定的能力。

(3) 易用性

① 易理解性

易理解性指软件产品使用户能理解软件是否合适以及如何能将软件用于特定的任务和使用环境的能力。

② 易学性

易学性指软件产品使用户能学习它的能力。

③ 易操作性

易操作性指软件产品使用户能操作和控制它的能力。

④ 吸引性

吸引性指软件产品吸引用户的能力。

⑤ 易用依从性

易用依从性指软件产品依附于同易用性相关的标准、约定、风格指南或规定的能力。

(4) 效率

① 时间特性

时间特性指在规定条件下,软件产品执行其功能时,提供适当的响应和处理时间以及吞吐率的能力。

② 资源利用性

资源利用性指在规定条件下,软件产品执行其功能时,使用合适的数量和类型的资源的能力。

③ 效率依从性

效率依从性指软件产品依附于同效率相关的标准或约定的能力。

(5) 维护性

维护性指软件产品可被修改的能力。修改可能包括修正、改进或软件适应环境、需求和功能规格说明中的变化。

① 易分析性

易分析性指软件产品诊断软件缺陷或失效原因,以及判定待修改的部分的能力。

② 易改变性

易改变性指软件产品使指定的修改可以被实现的能力。

③ 稳定性

稳定性指软件产品避免由于软件修改而造成意外结果的能力。

④ 易测试性

易测试性指软件产品使已修改软件能被确认的能力。

⑤ 维护依从性

维护依从性指软件产品依附于同维护性相关的标准或约定的能力。

(6) 可移植性

① 适应性

适应性指软件产品无须采用有别于为考虑该软件的目的而准备的活动或手段,就可能适应不同的指定环境的能力。

② 易安装性

易安装性指软件产品在指定环境中被安装的能力。

③ 共存性

共存性指软件产品在公共环境中同与其分享公共资源的其他独立软件共存的能力。

④ 易替换性

易替换性指软件产品在环境相同、目的相同的情况下替代另一个指定软件产品的能力。

⑤ 可移植性依从性

可移植性依从性指软件产品依附于同可移植性相关的标准或约定的能力。

6) 使用质量的质量模型

使用质量是从用户角度看待的质量,其属性分为 4 种,即有效性、生产率、安全性和满意度。使用质量的质量模型如图 2-17 所示。

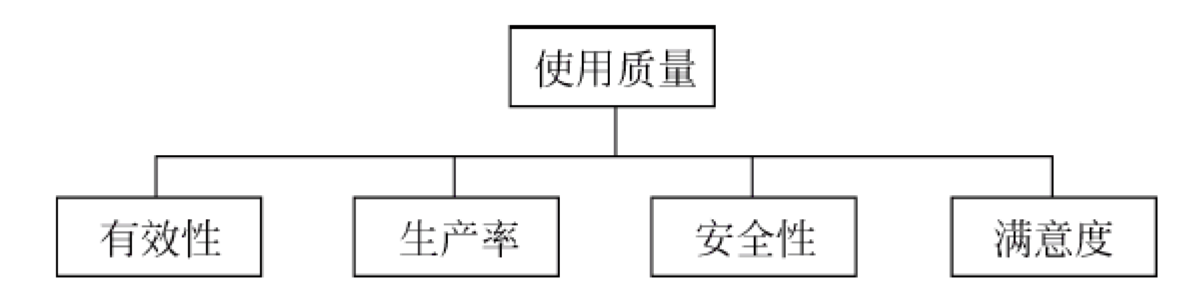


图 2-17 使用质量的质量模型

(1) 有效性

有效性指软件产品在指定的环境下,使用户获得满足准确度和完整性要求的规定目标的能力。

(2) 生产率

生产率指软件产品在指定的使用环境下,使用户可使用与获得的有效性有关的合适数量资源的能力。

(3) 安全性

安全性指软件产品在指定使用环境下,获得可接受的对人类、事务、软件、财产或环境有害的风险级别的能力。

(4) 满意度

满意度指软件产品在指定使用环境下使用户满意的能力。

2.5 软件质量与软件测试

软件质量是贯穿软件生存期的一个极为重要的问题,是软件开发过程中所使用的各种开发技术和验证方法的最终体现。因此,在软件生存期中要特别重视质量的保证,以生成高质量的软件产品。

软件质量是一个软件企业成功的必要条件,其重要性无论怎样强调都不过分。软件质量与传统意义上的质量概念并无本质差别,只是针对软件的某些特性进行了调整。

软件质量由以下三部分构成。

(1) 软件产品的质量,即满足使用要求的程度。

(2) 软件开发过程的质量,即能否满足开发所带来的成本、时间和风险等要求。

(3) 软件在其商业环境中所表现的质量。

总结起来,高品质软件应该是相对的无产品缺陷或只有极少量的缺陷,它能够准时递交给客户,所花费用都在预算内,并且满足客户需求,是可维护的。但是,有关质量好坏的最终评价依赖于用户的反馈。

软件质量具有以下三个特性。

(1) 可说明性:用户可以基于产品或服务的描述和定义加以使用。

(2) 有效性:产品或服务对于客户的需求是否能保持有效,如具有 99.99%有效性,可以说达到质量要求。

(3) 易用性:对于用户、产品或服务非常容易使用并且一定是非常有用的功能。

软件质量正确的内涵包括三方面,完整的需求、正确的代码和最少的缺陷。软件测试就是在软件开发的整个生命周期中对这三方面进行有效控制的重要手段,是软件质量的安全副驾驶。据统计,通过必要测试,软件缺陷数可至少降低 75%,而软件的投资回报率能达到 350%。

然而,在软件测试的早期,人们却远远没有意识到其对质量控制的重要性。那时的测试等同于“调试”,目的仅仅是纠正软件中已经知道的故障,常由开发人员自己完成,对测试的精力投入较少,介入时间也较晚。随着 IT 业的蓬勃发展,软件越来越趋向大型化、高复杂度,软件质量问题逐步加剧,软件测试开始被企业所重视。软件测试的过程也由最初的后期调试上升到软件开发全生命周期的质量控制,测试的重要性和规范性也不断提高。在信产部关于计算机系统集成资质及信息系统工程监理资质的认证中,软件测试能力已被定为评价公司技术能力的一项重要指标。“以测代评”也成为我国科技项目择优支持的一项重要举

措，据悉国家“863”计划对数据库管理系统、操作系统、办公软件等项目的经费支持，都要通过专业机构的测试结果来决定。

除此以外，软件测试的方式也开始由手工向自动化测试方向转变，测试工具包括白盒、黑盒、嵌入式等七大类。软件测试的蓬勃发展在促进质量提升的同时，也将软件产业带进发展的新时代。信产部信息产品管理司处长孙文龙曾经强调，信产部将把软件产品的功能测试作为下阶段发展的重要内容，而培养专业的人才又是重中之重。

把软件质量依赖于测试，是不可能真正解决软件质量问题的。

测试不是解决错误的根本举措，只是一种辅助手段。但又是必需的手段。

软件测试的首要任务是发现错误。发现错误也许要花很大的代价。因为测试是复杂的，不存在好的办法使每次测试都是有效的。有这么一句话：如果做某件事有两种或多种方法，其中有一种方法会导致一个灾难结局，那么也会有另一种方法导致同样的结局。

测试的复杂性和软件的复杂性是一致的。也就是说由于软件的复杂导致了测试的复杂。

测试提出了基本的和令人困惑的难题。假使我们在测试时从来不希望检测被测系统所有可能的输入、路径和状态，那么应该选择什么？什么时候应该停止？如果我们必须依赖于测试来防止某种失败，那么我们怎么来设计既是可测试的又是有效的系统呢？我们怎样来编写一个测试包，它可以检测足够多的消息和状态的组合来说明没有失败的操作，但是从实用性来说它又足够的小？

其次对于给定的测试包，说明被测系统是符合规约所描述的需求。

所以，测试活动与软件过程协调一致得好与坏，都对测试的有效性有很重要的影响。

如果测试的开发是在一个项目开始时进行的，那么测试将是非常有效的。及早考虑可测试性，及早进行测试设计，和尽可能早地实现测试，提高了有力预防错误的方法。这些过程迫使设计人员更仔细地考虑需求和规约的实现，其结果可能改进应用设计。关于体系结构、详细设计和编码实践的及早决策，都能使测试变得更容易更经济。

软件质量是指软件产品的特性可以满足用户的功能、性能需求的能力。软件过程是人们通常所说的软件生命周期中的活动，一般包括软件需求分析、软件设计、软件编码、软件测试、交付、安装和软件维护。随着软件过程的开始，软件质量也逐减建立起来。软件过程的优劣决定了软件质量的高低，好的过程是高效高质量的前提。人员和过程是决定软件质量的关键因素。高质量的人员和好的过程应该得到好的产品。

软件系统的开发包括一系列生产活动，其中由人带来的错误因素非常多，错误可能出现在程序的最初需求分析阶段，设计目标可能是错误的或描述不完整，也可能在后期的设计和开发阶段，因为人员之间的交流不够，交流上有误解或者根本不进行交流，所以尽管人们在开发软件的过程中使用了许多保证软件质量的方法和技术，单开发出的软件中还会隐藏许多错误和缺陷。可见，只有通过严格的软件测试，才能很好地提高软件质量，而软件质量并不是依靠软件测试来保证的，软件的质量要靠不断地提高技术水平和改进软件开发过程来保证，软件测试只是一种有效地提高软件质量的技术手段，而不是软件质量的安全网。

软件测试能够找出软件缺陷，确保软件产品满足需求。但是测试不是质量保证，二者并不等同。测试可以查找错误并进行修改，从而提高软件产品的质量。软件测试避免错误以求高质量，并且还有其他方面的措施以保证质量问题，如软件质量保证。

正规的软件测试系统主要包括制定测试计划、测试设计、实施测试、建立和更新测试文档。而软件质量保证的工作主要为：制定软件质量要求，组织正式度量，软件测试管理，对软件的变更进行控制，对软件质量进行度量，对软件质量情况及时记录和报告。软件质量保证的职能是向管理层提供正确的可行信息，从而促进和辅助设计流程的改进。软件质量保证的职能还包括监督测试流程，这样测试工作就可以被客观地审查和评估，同时也有助于测试流程的改进。二者的不同之处在于软件质量保证工作侧重对软件开发流程中的各个过程进行管理与控制，杜绝软件缺陷的产生。而测试则是对已产生的软件缺陷进行修复。

软件测试对软件质量的影响：

由于人们对于软件质量的重视程度越来越高，就导致了软件测试在软件开发中的地位越来越重要。软件测试是程序的一种执行过程，目的是尽可能发现并改正被测试软件中的错误，提高软件的可靠性。它是软件生命周期中一项很重要且非常复杂的工作，对软件可靠性保证具有极其重要的意义。

在目前形式化方法和程序正确性证明技术还无望成为实用性方法的情况下，软件测试在将来相当一段时间内仍然是软件可靠性保证的有效方法。软件工程的总目标是充分利用有限的人力和物力资源，高效率、高质量地完成软件开发项目。不足的测试势必使软件带着一些未揭露的隐藏错误投入运行，这将意味着更大的危险让用户承担，过度测试则会浪费许多宝贵的资源。到测试后期，即使找到了错误，然而付出了过高的代价。E. W. Dijkstra 的一句名言说明了这一道理：“程序测试只能表明错误的存在，而不能表明错误不存在。”可见，测试是为了使软件中蕴涵的缺陷低于某一特定值，使产出、投入比达到最大。

软件测试和软件质量是分不开的。测试是手段，质量是目的。对比国外可以看到，国外软件开发机构会把 40%的工作花在测试上，测试费用则会占到软件开发总费用的 30%到 50%，对于一些要求高可靠性、高安全性的软件，测试费用则相当于整个软件项目开发费用的 3 至 5 倍。因此，软件测试在软件生存期中占有非常突出的位置，是保证软件质量的重要手段。

软件项目的实践一再说明，为了确保软件产品能够符合用户的需要，必须着眼于整个软件生存周期，在各个阶段进行验证、确认和测试活动，使软件不致在开发完成后，才发现和用户的需求有较大的差距。

2.6 软件质量保证与软件测试

软件质量保证(Software Quality Assurance,SQA)是建立一套有计划、有系统的方法，来向管理层保证拟定出的标准、步骤、实践和方法能够正确地被所有项目所采用。软件质量保证的目的是使软件过程对于管理人员来说是可见的。它通过对软件产品和活动进行评审和审计来验证软件是合乎标准的。软件质量保证所要实现的主要目标包括：软件质量保证工作是有计划进行的；客观地验证软件项目产品和工作是否遵循恰当的标准、步骤和需求；将软件质量保证工作及结果通知给相关组别和个人；高级管理层接触到在项目内部不能解决的不符合类问题；软件质量需要全面的测试工作来保证。

软件质量保证的目的不是为了保证产品质量，保证产品质量是软件测试的任务。软件质量保证主要是提供确信。因此要对了解客户要求开始至售后服务的全过程进行管理。这

就要求企业建立品管体系，制定相应的文件规范各过程的活动并留下活动实施的证据，以便提供信任。

软件测试和软件质量保证的主要区别是前者是保证产品质量符合规定，后者是建立体系并确保体系按要求运作，以提供内外部的信任。同时软件测试和软件质量保证又有相同点：即软件测试和软件质量保证都要进行验证，如软件测试按标准检测产品，就是验证产品是否符合规定要求，软件质量保证进行内审就是验证体系运作是否符合标准要求。

软件测试人员的职责是尽可能早地发现缺陷，并确保缺陷得以修复。软件质量保证人员的职责是创建和执行改进软件开发过程，并防止软件缺陷发生的标准和方法。一个关注如何避免产生缺陷，侧重点是控制产品产生的过程；一个关注如何尽快发现缺陷，重点是产品本身。

软件质量保证活动与测试的关系如图 2-18 所示。

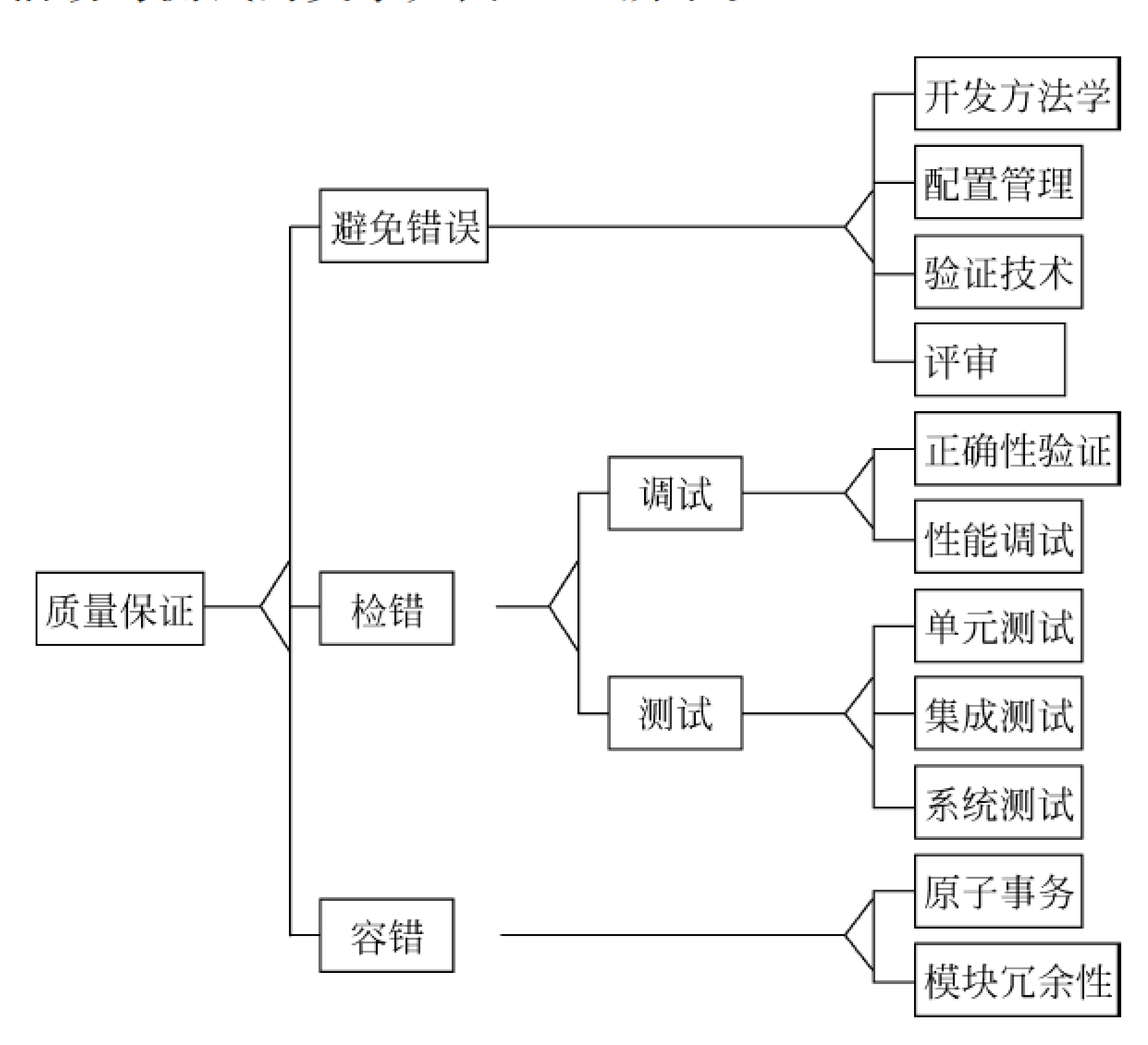


图 2-18 软件质量保证活动与测试的关系

质量保证的主要工作范围如下。

（1）指导并监督项目按照过程实施。

（2）对项目进行度量、分析，增加项目的可视性。

（3）审核工作产品，评价工作产品和过程质量目标的符合度。

（4）进行缺陷分析，缺陷预防活动，发现过程的缺陷，提供决策参考，促进过程改进。

质量保证和测试的关系如下。

（1）SQA 从流程方面保证软件的质量。

（2）测试从技术方面保证软件的质量。

软件测试和软件质量保证是软件质量工程的两个不同层面的工作。软件测试只是软件质量保证工作的一个重要环节。

质量保证的工作是通过预防、检查和改进来保证软件质量。QA 采取的方法主要是按照“全面质量管理”和“过程管理并改进”的原则来展开工作。在质量保证的工作中会掺入一些测试活动，但它所关注的是软件质量的检查和测量。因此，其主要工作是着眼于软件开发

活动中的过程、步骤和产物，并不是对软件进行剖析，找出问题和评估。

测试虽然也与开发过程紧密相关，但它所关心的不是过程的活动，相对的是关心结果。测试人员要对过程中的产物（开发文档和源代码）进行静态审核，运行软件，找出问题，报告质量甚至评估，而不是为了验证软件的正确性。当然，测试的目的是为了去证明软件有错，否则就违背了测试人员的本职。因此，测试虽然对提高软件质量起了关键的作用，但它只是软件质量保证中的一个重要环节。

很少有人从非技术角度去分析这两者的区别，但本书作者认为，从公司业务出发，QA的工作是相对前置的，并可能含有某种公关性质；而软件测试相对后置，是内部层面的工作。这也同样验证了两者的本质区别，即“软件测试和软件质量保证是软件质量工程的两个不同层面的工作。软件测试只是软件质量保证工作的一个重要环节。”

软件质量保证：SQA介入于整个软件开发过程-监督和改进过程，确认达成的标准和过程被正确地遵循，保证问题被发现和解决。它以预防为主。

软件测试：软件测试是在一定控制的条件下，围绕一个系统或应用的操作并且评价其结果（一个简单的例子：如果用户使用硬件A，在一个应用接口B上做了操作C，那么结果D应当出现），控制的条件应当包括正常和异常的条件。测试企图使事情变得很糟，从而来检测出一些应当发生而没发生，或者不应该发生而发生的事情。测试以检测为主。

软件测试人员的一项重要任务是提高软件质量，但不等于说软件测试人员就是软件质量保证人员，因为测试只是质量保证工作中的一个环节。软件质量保证和软件测试是软件质量工程的两个不同层面的工作。

思考题

1. 质量的定义是什么？
2. 软件质量的定义是什么？
3. 影响软件质量的主要因素有哪些？
4. 常见的软件质量模型有哪几种？它们各有什么特点？
5. 质量标准的发展情况是怎样的？
6. 软件质量与软件测试、软件质量保证与软件测试之间有什么联系？

技　术　篇

本篇内容：

- 面向传统开发过程的软件测试
- 面向软件工程层面的软件测试
- 自动化测试
- 敏捷测试

第3章 面向传统开发过程的软件测试

本章学习重点

- 熟悉常见的软件测试模型以及它们之间的区别。
- 掌握软件生命周期的几个阶段。
- 了解常见的软件开发模型。
- 了解什么是单元测试、集成测试和系统测试。
- 熟悉单元测试、集成测试和系统测试的主要工作内容。
- 掌握编写测试用例的方法。

本章学习难点

熟悉常见软件测试模型之间的区别。

3.1 软件测试模型

目前常见的主流软件生命周期模型或软件开发过程模型有瀑布模型、原型模型、螺旋模型、增量模型、渐进模型、快速软件开发(RAD)以及 Rational 统一过程(RUP)等,这些模型对于软件开发过程具有很好的指导作用。但在这些过程方法中,软件测试的地位和价值并没有体现出来,也没有给软件测试以足够的重视,利用这些模型显然无法更好地指导测试实践。软件测试是与软件开发紧密相关的一系列有计划、系统性的活动,软件测试也需要测试模型去指导实践。下面先对主要的软件测试模型做一些简单的介绍,再补充软件生命周期做介绍。

1. V 模型

V 模型是最具有代表性的测试模型。V 模型最早是由 Paul Rook 在 20 世纪 80 年代后期提出的,V 模型在英国国家计算中心文献中发布,旨在改进软件开发的效率和效果。

在传统的开发模型中,例如瀑布模型,通常把测试过程作为在需求分析、概要设计、详细设计和编码全部完成之后的一个阶段,尽管有时测试工作会占用整个项目周期一半的时间,但是有人仍认为测试只是一个收尾工作,而不是主要的工程。V 模型的推出就是对此种认

识的改进。V 模型是软件开发瀑布模型的变种，它反映了测试活动与分析和设计的关系，从左到右，描述了基本的开发过程和测试行为，明确地标明了测试工程中存在的不同级别，清楚地描述了这些测试阶段和开发过程期间各阶段的对应关系，如图 3-1 所示。

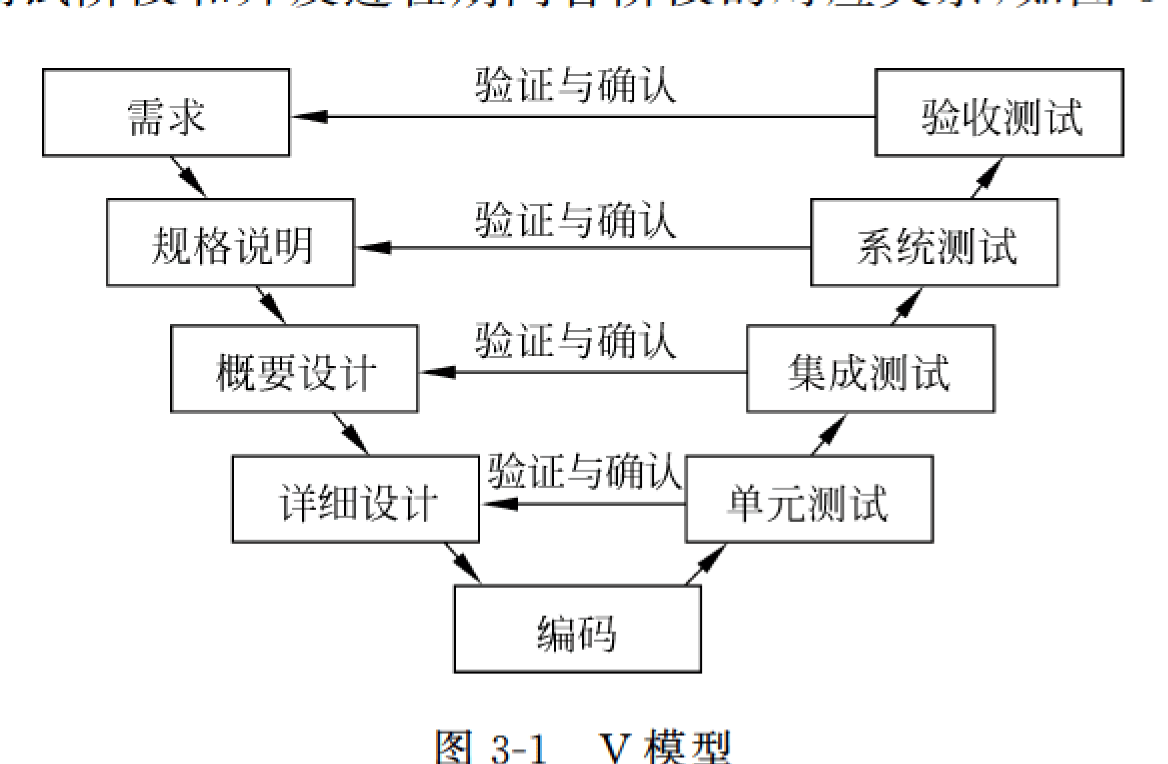


图 3-1　V 模型

图 3-1 中，V 模型图中箭头代表了时间方向，左边下降的是开发过程各阶段，与此相对应的是右边上升的部分，即测试过程的各个阶段。

V 模型的软件测试策略既包括底层测试又包括上层测试，底层测试是为了确保源代码的正确性，上层测试是为了使整个系统满足用户的需求。

V 模型指出，单元和集成测试是验证程序设计，开发人员和测试组应检测程序的执行是否满足软件设计的要求；系统测试应当验证系统设计，检测系统功能、性能的质量特性是否达到系统设计的指标；由测试人员和用户进行软件的确认测试和验收测试，追溯软件需求说明书进行测试，以确定软件的实现是否满足用户需求或合同的要求。

V 模型存在一定的局限性，它仅把测试过程作为在需求分析、概要设计、详细设计及编码之后的一个阶段。容易使人理解为测试是软件开发的最后一个阶段，主要是针对程序进行测试寻找错误，而需求分析阶段隐藏的问题一直到后期的验收测试才被发现。

类比记忆：此模型与软件开发模式中的线性模型(典型的瀑布模型)有相似的不足，在瀑布模型中，测试阶段处于软件实现后，这意味着必须在代码完成后有足够的时间预留给测试活动，否则将导致测试不充分，开发前期未发现的错误会传递并扩散到后面的阶段，而在后面发现这些错误时，可能已经很难回头再修正，从而导致项目的失败。

2. W 模型

V 模型的局限性在于没有明确地说明早期的测试，不能体现“尽早地和不断地进行软件测试”的原则。在 V 模型中增加软件各开发阶段应同步进行的测试，被演化为一种 W 模型，因为实际上开发是“V”，测试也是与此相并行的“V”。只不过把这二者结合起来进行“并行工程”而已，基于“尽早地和不断地进行软件测试”的原则，在软件的需求和设计阶段的测试活动应遵循 IEEE STD 1012—1998《软件验证和确认(V&V)》的原则。

一个基于 V&V 原理的 W 模型示意图如图 3-2 所示。

相对于 V 模型，W 模型更科学。W 模型可以说是 V 模型自然而然的发展。W 模型强调测试伴随着整个软件开发周期，而且测试的对象不仅是程序，需求、功能和设计同样要测试。这样，只要相应地开发活动完成，就可以开始执行测试，可以说，测试与开发是同步进行

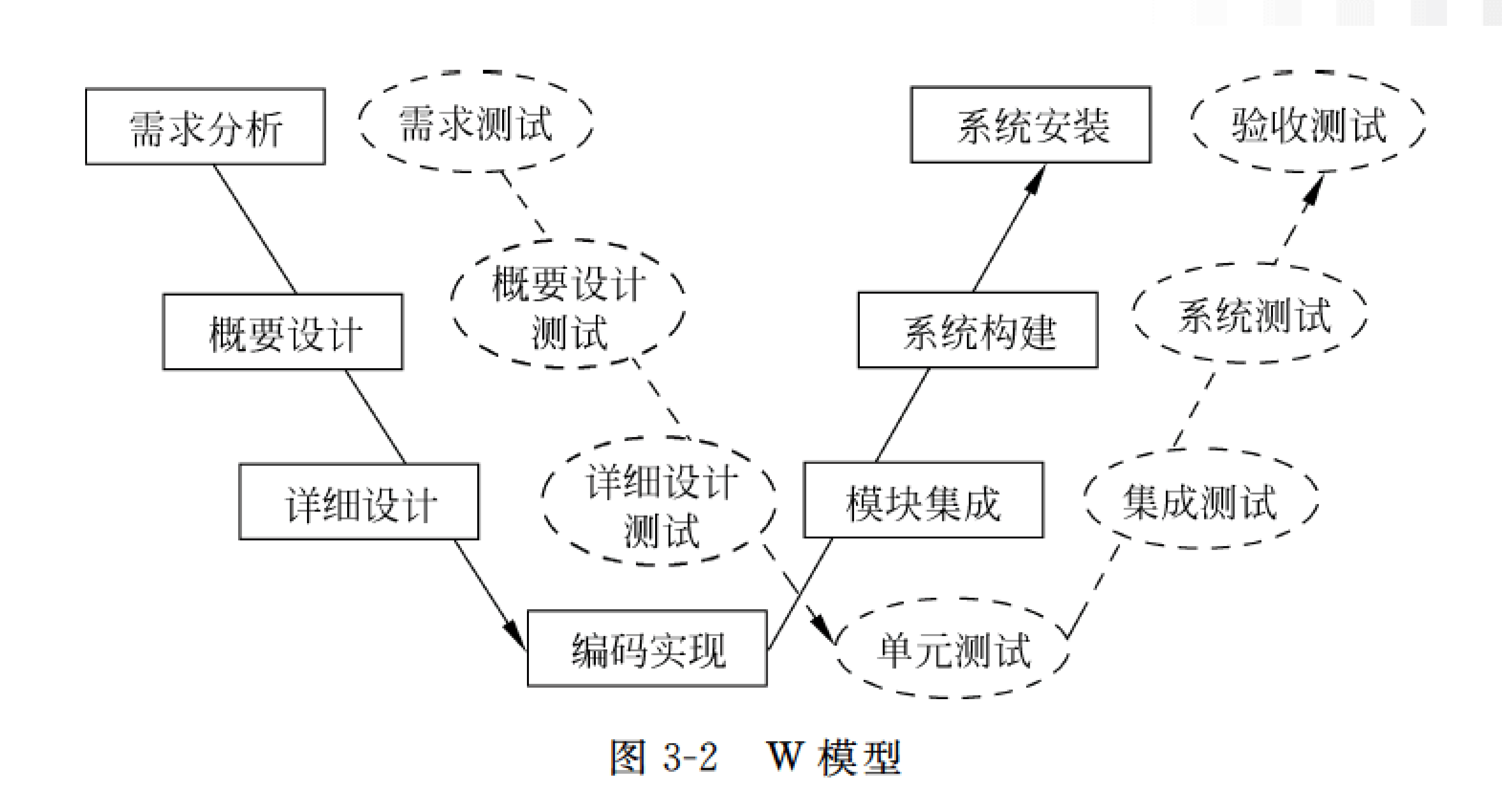


图 3-2　W 模型

的，从而有利于尽早地发现问题。以需求为例，需求分析一完成，就可以对需求进行测试，而不是等到最后才进行针对需求的验收测试。

如果测试文档能尽早提交，那么就有了更多的检查和检阅的时间，这些文档还可用于评估开发文档。另外还有一个很大的益处是，测试者可以在项目中尽可能早地面对规格说明书中的挑战。这意味着测试不仅是评定软件的质量，还可以尽可能早地找出缺陷所在，从而帮助改进项目内部的质量。参与前期工作的测试者可以预先估计问题和难度，这将可以显著地减少总体测试时间，加快项目进度。

根据 W 模型的要求，一旦有文档提供，就要及时确定测试条件，以及编写测试用例，这些工作对测试的各级别都有意义。当需求被提交后，就需要确定高级别的测试用例来测试这些需求。当概要设计编写完成后，就需要确定测试条件来查找该阶段的设计缺陷。

W 模型也是有局限性的。这一点实际上源于 V 模型的缺陷，W 模型和 V 模型都把软件的开发视为需求、设计、编码等一系列串行的活动。同样，软件开发和测试保持一种线性的前后关系，需要有严格的指令表示上一阶段完全结束，才可以正式开始下一个阶段。这样就无法支持迭代、自发性以及变更调整。对于当前很多文档需要事后补充，或者根本没有文档的做法（这已成为一种开发的文化），开发人员和测试人员都面临同样的困惑。

类比记忆：W 模型相当于两个 V 模型的叠加，一个是开发的 V，一个是测试的 V，由于在项目中开发和测试是同步进行的，相当于两个 V 是并列同步进行的，测试在一定程度是随着开发的进展而不断向前进行的。

3. H 模型

V 模型和 W 模型均存在一些不妥之处。首先，如前所述，它们都把软件的开发视为需求、设计、编码等一系列串行的活动，而事实上，虽然这些活动之间存在相互牵制的关系，但在大部分时间内，它们是可以交叉进行的。虽然软件开发期望有清晰的需求、设计和编码阶段，但实践告诉我们，严格的阶段划分只是一种理想状况。试问，有几个软件项目是在有了明确的需求之后才开始设计的呢？所以，相应的测试之间也不存在严格的次序关系。同时，各层次之间的测试也存在反复触发、迭代和增量关系。其次，V 模型和 W 模型都没有很好地体现测试流程的完整性。

为了解决以上问题，提出了 H 模型。它将测试活动完全独立出来，形成一个完全独立的流程，将测试准备活动和测试执行活动清晰地体现出来。H 模型的简单示意图如图 3-3 所示。

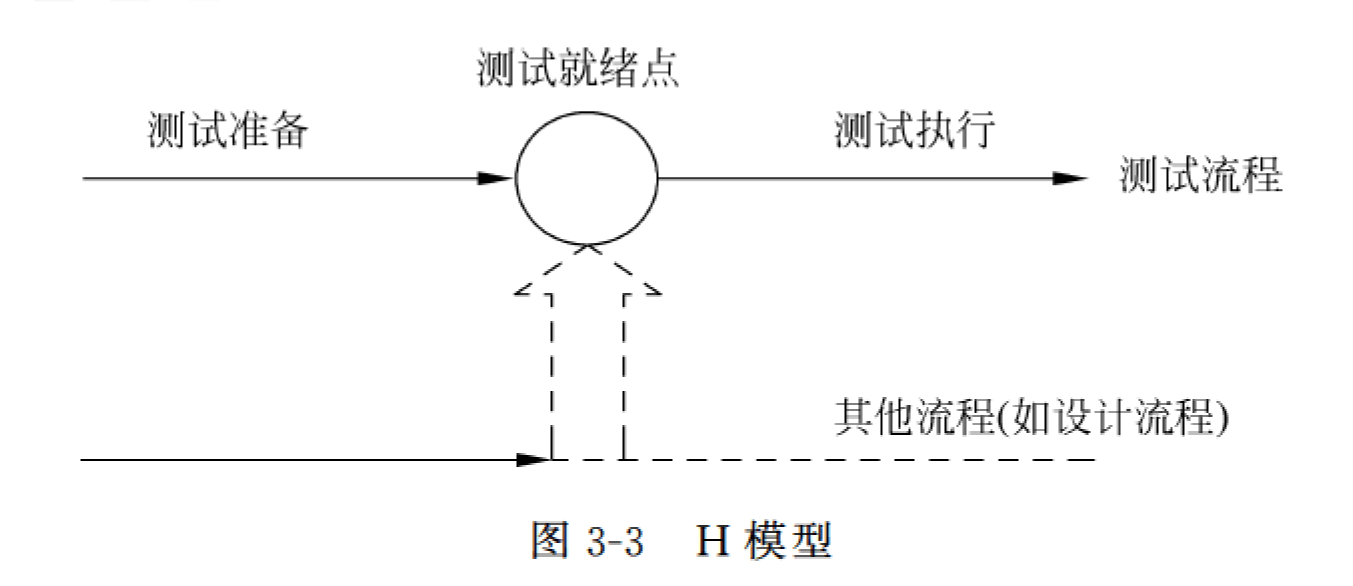


图 3-3 H 模型

H 模型图仅演示了在整个生存周期中某个层次上的一次测试"微循环"。图中的其他流程可以是任意开发流程。例如，设计流程和编码流程。也可以是其他非开发性流程，例如 SQA 流程，甚至是测试流程自身。也就是说，只要测试条件成熟，测试准备活动完成了，测试执行活动就可以(或者说需要)进行了。

概括地说，H 模型揭示了：

(1) 软件测试不仅指测试的执行，还包括很多其他的活动。

(2) 软件测试是一个独立的流程，贯穿产品整个生命周期，与其他流程并发地进行。

(3) 软件测试要尽早准备，尽早执行。

(4) 软件测试是根据被测物的不同而分层次进行的。不同层次的测试活动可以是按照某个次序先后进行的，但也可能是反复的。

在 H 模型中，软件测试模型是一个独立的流程，贯穿于整个产品周期，与其他流程并发地进行。当某个测试时间点就绪时，软件测试即从测试准备阶段进入测试执行阶段。

4. X 模型

下面介绍另外一种测试模型，即 X 模型，其目标是弥补 V 模型的一些缺陷。该模型也是对 V 模型的改进，X 模型提出针对单独的程序片段进行相互分离的编码和测试，此后通过频繁的交接，通过集成最终合成为可执行的程序。软件测试 X 模型如图 3-4 所示。

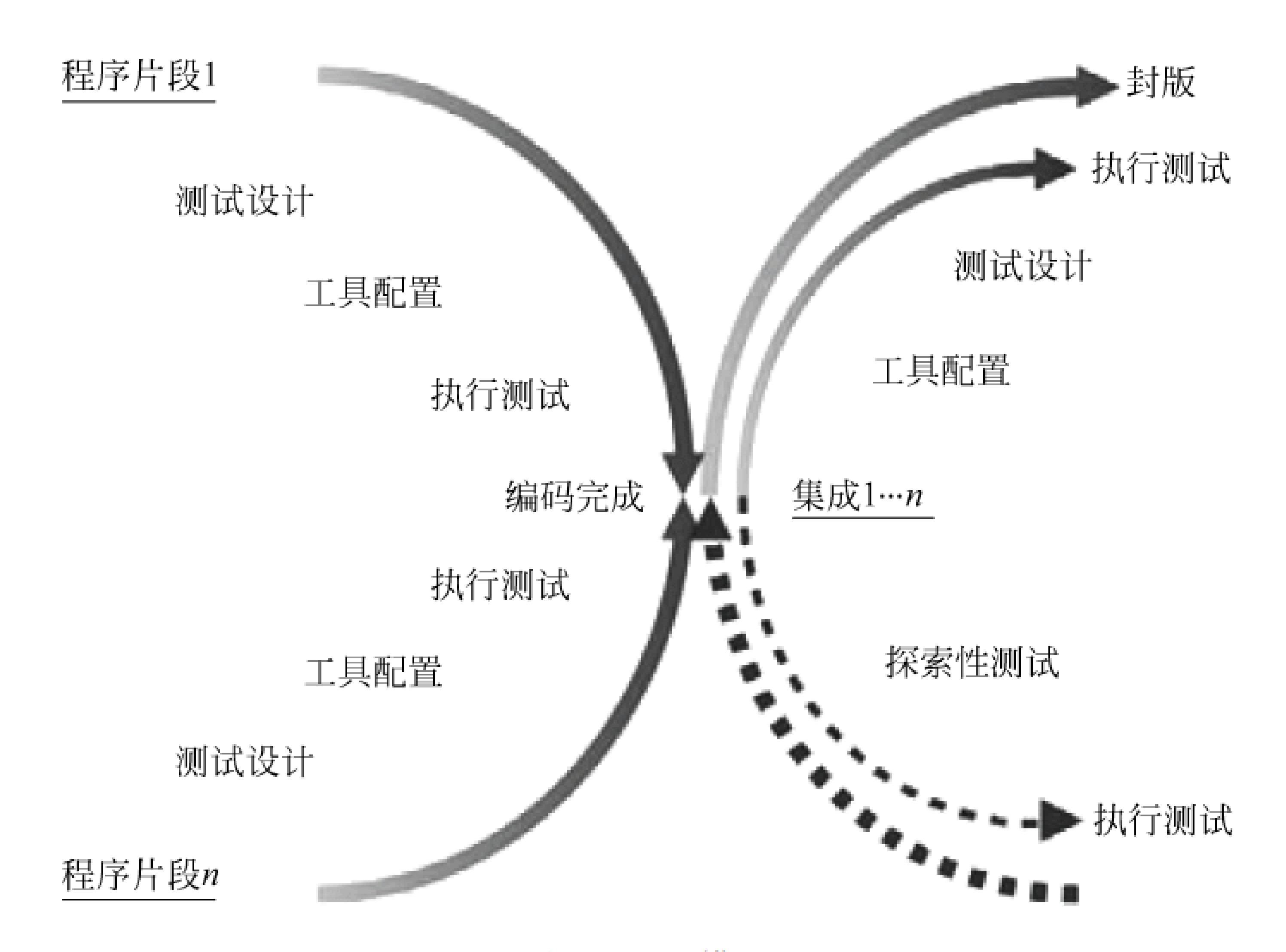


图 3-4 X 模型

X 模型的基本思想是由 Marick 提出的，Marick 对 V 模型最主要的批评是 V 模型无法引导项目的全部过程。他认为一个模型必须能处理开发的所有方面，包括交接、频繁重复的集成以及需求文档的缺乏等。Marick 认为一个模型不应该规定那些和当前所公认的实践不一致的行为。

X 模型的左边描述的是针对单独程序片段所进行的相互分离的编码和测试，此后将进行频繁的交接，通过集成最终成为可执行的程序，然后再对这些可执行程序进行测试。已通过集成测试的成品可以进行封装并提交给用户，也可以作为更大规模和范围内集成的一部分。多根并行的曲线表示变更可以在各个部分发生。从图 3-4 中可见，X 模型还定位了探索性测试，这是不进行事先计划的特殊类型的测试，这一方式往往能帮助有经验的测试人员在测试计划之外发现更多的软件错误。但这样可能对测试造成人力、物力和财力的浪费，对测试员的熟练程度要求比较高。

Marick 对 V 模型提出质疑，也是因为 V 模型是基于一套必须按照一定顺序严格排列的开发步骤，而这很可能并没有反映实际的实践过程。因为在实践过程中，很多项目是缺乏足够的需求的，而 V 模型还是从需求处理开始。

Marick 也质疑了单元测试和集成测试的区别，因为在某些场合人们可能会跳过单元测试而热衷于直接进行集成测试。Marick 担心人们盲目地跟随“学院派的 V 模型”，按照模型所指导的步骤进行工作，而实际上某些做法并不切合实际。

5. 前置测试模型

前置测试模型是将测试和开发紧密结合的模型，该模型提供了轻松的方式，可以使项目加快速度。

前置测试模型如图 3-5 所示。

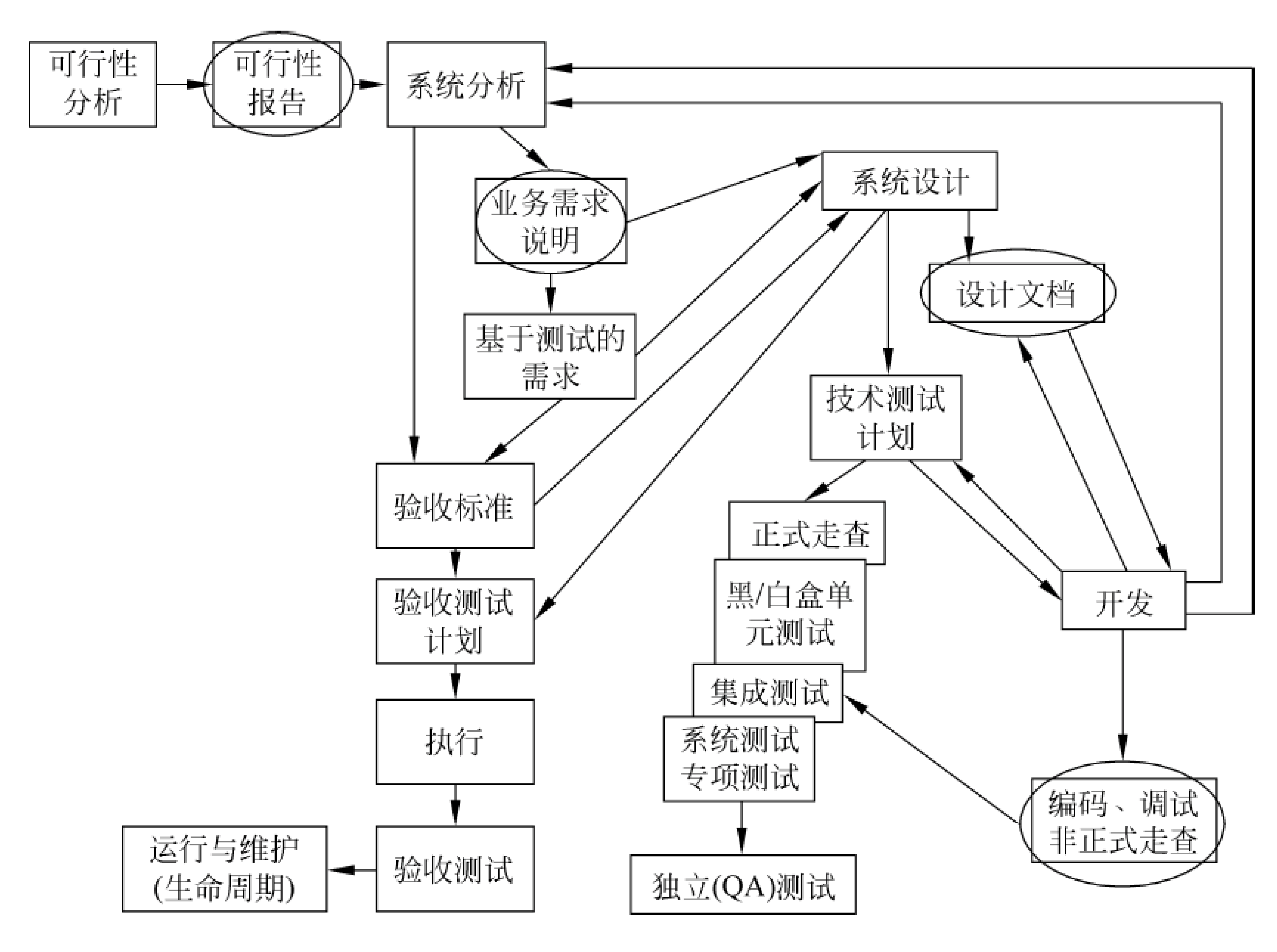


图 3-5 前置测试模型

前置测试模型体现了以下要点。

(1) 开发和测试相结合。前置测试模型将开发和测试生命周期整合在一起,标识了项目生命周期从开始到结束之间的关键行为,并且标识了这些行为在项目周期中的价值所在。如果其中有些行为没有得到很好的执行,那么项目成功的可能性就会因此而有所降低。如果有业务需求,则系统开发过程将更有效率。我们认为在没有业务需求的情况下进行开发和测试是不可能的。而且,业务需求最好在设计和开发之前就被正确定义。

(2) 对每一个交付内容进行测试。每一个交付的开发结果都必须通过一定的方式进行测试。源程序代码并不是唯一需要测试的内容。图中的椭圆框表示了其他一些要测试的对象,包括可行性报告、业务需求说明,以及系统设计文档等。这同 V 模型中开发和测试的对应关系是一致的,并且在其基础上有所扩展,变得更为明确。

(3) 在设计阶段进行测试计划和测试设计。设计阶段是做测试计划和测试设计的最好时机。很多组织要么根本不做测试计划和测试设计,要么在即将开始执行测试之前才飞快地完成测试计划和测试设计。在这种情况下,测试只是验证了程序的正确性,而不是验证整个系统本该实现的东西。

(4) 测试和开发结合在一起。前置测试将测试执行和开发结合在一起,并在开发阶段以编码—测试—编码—测试的方式来体现。也就是说,程序片段一旦编写完成,就会立即进行测试。一般情况下,先进性的测试是单元测试,因为开发人员认为通过测试来发现错误是最经济的方式。但也可参考 X 模型,即一个程序片段也需要相关的集成测试,甚至有时还需要一些特殊测试。对于一个特定的程序片段,其测试的顺序可以按照 V 模型的规定,但其中还会交织一些程序片段的开发,而不是按阶段完全地隔离。

(5) 让验收测试和技术测试保持相对独立。验收测试应该独立于技术测试,这样可以提供双重的保险,以保证设计及程序编码能够符合最终用户的要求。验收测试既可以在实施的第一步来执行,也可以在开发阶段的最后一步执行。前置测试模型提倡验收测试和技术测试沿循两条不同的路线来进行,每条路线分别验证系统是否能够如预期设想的那样进行正常工作。这样,当单独设计好的验收测试完成了系统的验证时,即可确信这是一个正确的系统。

前置测试模型包括两项测试计划技术,这也是需求测试技术中的一部分。其中的第一项技术是开发基于需求的测试用例。这并不仅仅是为以后提交上来的程序的测试做好初始化准备,也是为了验证需求是否是可测试的。这些测试可以交由用户来进行验收测试,或者由开发部门做某些技术测试。很多测试团体都认为,需求的可测试性即使不是需求首要的属性,也应是其最基本的属性之一。因此,在必要的时候可以为每一个需求编写测试用例。不过,基于需求的测试最多也只是和需求本身一样重要。一项需求可能本身是错误的,但它仍是可测试的。而且,无法为一些被忽略的需求来编写测试用例。

第二项技术是定义验收标准。在接受交付的系统之前,用户需要用验收标准来进行验证。验收标准并不仅仅是定义需求,还应在前置测试之前进行定义,这将帮助揭示某些需求是否正确,以及某些需求是否被忽略了。

在 V 模型中,验收测试最早被定义好,并在最后执行,以验证所交付的系统是否真正符合用户业务的需求。

与 V 模型不同的是,前置测试模型认识到验收测试中所包含的三种成分,其中的两种

都与业务需求定义相联系：即定义基于需求的测试，以及定义验收标准。但是，第三种则需要等到系统设计完成，因为验收测试计划是由针对按设计实现的系统来进行的一些明确操作定义所组成，这些定义包括：如何判断验收标准已经达到，以及基于需求的测试已算成功完成。

技术测试主要是针对开发代码的测试，例如，V 模型中所定义的动态的单元测试，集成测试和系统测试。另外，前置测试还提示我们应增加静态审查，以及独立的 QA 测试。

QA 测试通常跟随在系统测试之后，从技术部门的意见和用户的预期方面出发，进行最后的检查。同样的还有特别测试。将其取名为特别测试，并把该名称作为很多测试的一个统称，这些测试包括负载测试、安全性测试、可用性测试等，它们不是由业务逻辑和应用来驱动的。

对技术测试最基本的要求是验证代码的编写和设计的要求是否相一致。一致的意思是系统确实提供了要求提供的，并且系统并没有提供不要求提供的。技术测试在设计阶段进行计划和设计，并在开发阶段由技术部门来执行。

6. 测试模型的使用

前面介绍了几种典型的测试模型，应该说这些模型对指导测试工作的进行具有重要的意义，但任何模型都不是完美的。应该尽可能地去应用模型中对项目有实用价值的方面，但不强行地为使用模型而使用模型，否则也没有实际意义。

在这些模型中，V 模型强调了在整个软件项目开发中需要经历的若干个测试级别，而且每一个级别都与一个开发级别相对应，但它忽略了测试的对象不应该仅包括程序，或者说它没有明确地指出应该对软件的需求、设计进行测试，而这一点在 W 模型中得到了补充。W 模型强调了测试计划等工作的先行和对系统需求和系统设计的测试，但 W 模型和 V 模型一样也没有专门对软件测试流程予以说明，因为事实上，随着软件质量要求越来越为大家所重视，软件测试也逐步发展成为一个独立于软件开发的一系列活动，就每一个软件测试的细节而言，它都有一个独立的操作流程。例如，现在的第三方测试，就包含从测试计划和测试用例编写，到测试实施以及测试报告编写的全过程，这个过程在 H 模型中得到了相应的体现，表现为测试是独立的。也就是说，只要测试前提具备了，就可以开始进行测试。当然，X 模型和前置测试模型又在此基础上增加了许多不确定因素的处理情况，因为在真实项目中，经常会有变更的发生，例如，需要重新访问前一阶段的内容，或者跟踪并纠正以前提交的内容，修复错误，排除多余的成分，以及增加新发现的功能等。

因此，在实际的工作中，要灵活地运用各种模型的优点，在 W 模型的框架下，运用 H 模型的思想进行独立的测试，并同时将测试与开发紧密结合，寻找恰当的就绪点开始测试并反复迭代测试，最终保证按期完成预定目标。

3.2 软件生命周期

和任何事物一样，软件也有其孕育、诞生、成长、成熟和衰亡的生存过程，一般称其为“软件生命周期”。软件生命周期一般分为 6 个阶段，即制定计划、需求分析、设计、编码、测试、运行和维护。现实的软件开发的各个阶段之间的关系不可能是顺序且线性的，而应该是带有反馈的迭代过程。在软件工程中，这个复杂的过程用软件开发模型来描述和表示。

软件开发模型是跨越整个软件生存周期的系统开发、运行和维护所实施的全部工作和任务的结构框架，它给出了软件开发活动各阶段之间的关系。目前，常见的软件开发模型大致可分为如下三种类型。

(1) 以软件需求完全确定为前提的瀑布模型(Waterfall Model)。

(2) 在软件开发初始阶段只能提供基本需求时采用的渐进式开发模型，如螺旋模型(Spiral Model)。

(3) 以形式化开发方法为基础的变换模型(Transformational Model)。

1. 瀑布模型

瀑布模型是最早出现的软件开发模型，在软件工程中占有重要的地位，它提供了软件开发的基本框架。其过程是从上一项活动接收该项活动的工作对象作为输入，利用这一输入实施该项活动应完成的内容给出该项活动的工作成果，并作为输出传给下一项活动，如图 3-6 所示。

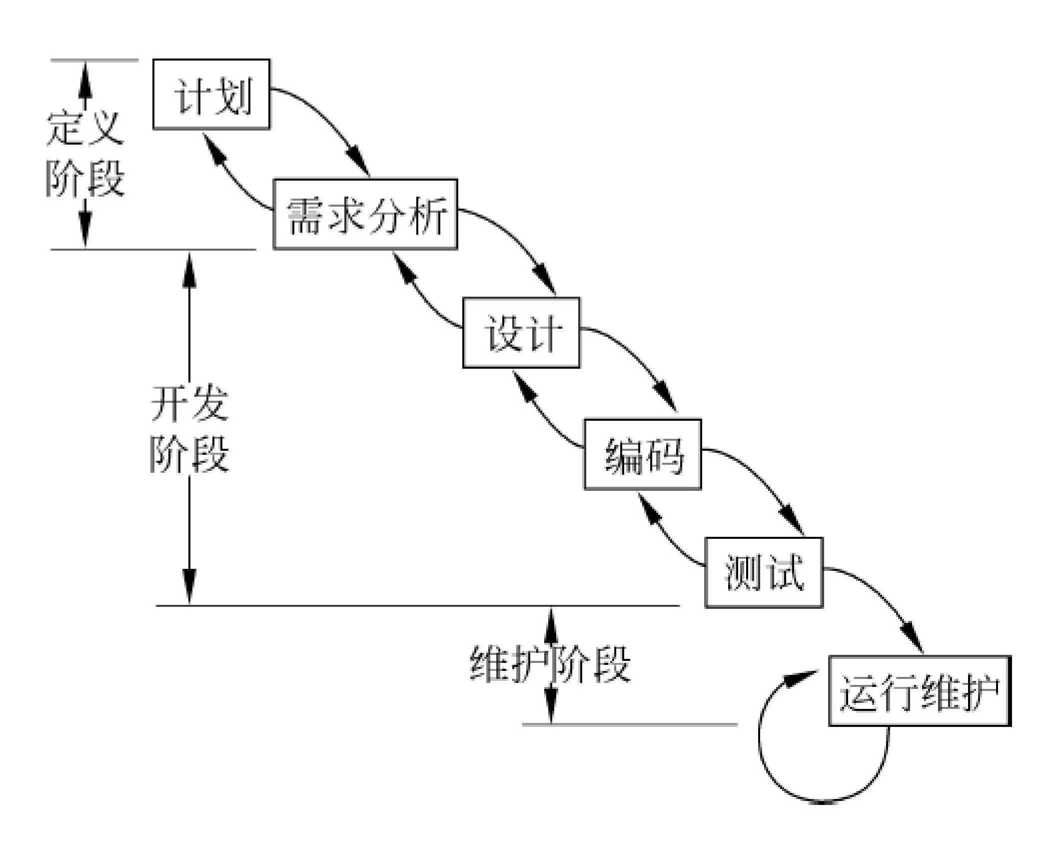


图 3-6 瀑布模型

瀑布模型核心思想是按工序将问题化简，将功能的实现与设计分开，便于分工协作，即采用结构化的分析与设计方法将逻辑实现与物理实现分开。将软件生命周期划分为制定计划、需求分析、软件设计、程序编写、软件测试和运行维护 6 个基本活动，并且规定了它们自上而下、相互衔接的固定次序，如同瀑布流水，逐级下落。

1) 瀑布模型的优点

(1) 为项目提供了按阶段划分的检查点。

(2) 当前一阶段完成后，只需要去关注后续阶段。

(3) 可在迭代模型中应用瀑布模型。

增量迭代应用于瀑布模型。迭代 1 解决最大的问题。每次迭代产生一个可运行的版本，同时增加更多的功能。每次迭代必须经过质量和集成测试。

2) 瀑布模型的缺点

(1) 在项目各个阶段之间极少有反馈。

(2) 只有在项目生命周期的后期才能看到结果。

(3) 通过过多的强制完成日期和里程碑来跟踪各个项目阶段。

2. 原型模型

原型模型如图 3-7 所示。

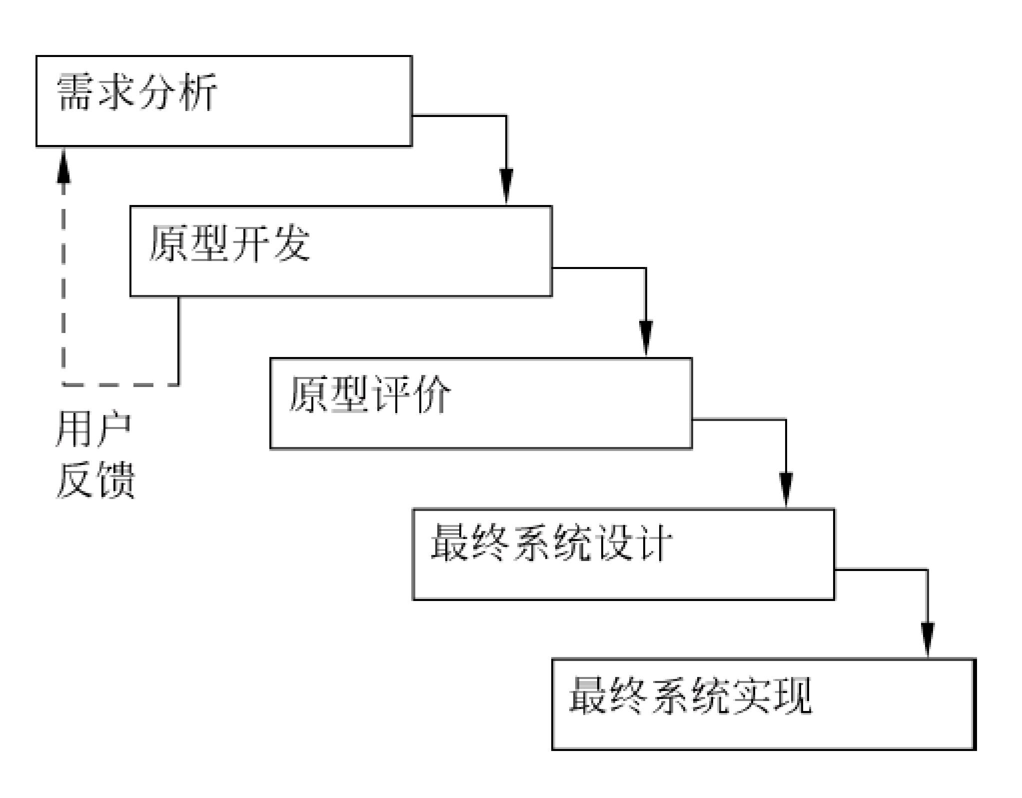


图 3-7 原型模型

原型模型的主要思想：先借用已有系统作为原型模型，通过“样品”不断改进，使得最后的产品就是用户所需要的。

原型模型通过向用户提供原型获取用户的反馈，使开发出的软件能够真正反映用户的需求。同时，原型模型采用逐步求精的方法完善原型，使得原型能够“快速”开发，避免了像瀑布模型一样在冗长的开发过程中难以对用户的反馈做出快速的响应。相对瀑布模型而言，原型模型更符合人们开发软件的习惯，是目前较流行的一种实用软件生存期模型。

1）原型模型的优点

（1）开发人员和用户在“原型”上达成一致。这样一来，可以减少设计中的错误和开发中的风险，也减少了对用户培训的时间，从而提高了系统的实用、正确性以及用户的满意程度。

（2）缩短了开发周期，加快了工程进度。

（3）降低成本。

2）原型模型的缺点

当告诉用户还必须重新生产该产品时，用户是很难接受的。这往往给工程继续开展带来不利因素。

不宜利用原型系统作为最终产品。采用原型模型开发系统，用户和开发者必须达成一致：原型被建造仅仅是用户用来定义需求，之后便部分或全部抛开，最终的软件是要充分考虑了质量和可维护性等方面之后才被开发出来。

3. 螺旋模型

螺旋模型采用一种周期性的方法来进行系统开发，这会导致开发出众多的中间版本。使用它，项目经理在早期就能够为客户实证某些概念。该模型是快速原型法，以进化的开发方式为中心，在每个项目阶段使用瀑布模型法。这种模型的每一个周期都包括需求定义、风险分析、工程实现和评审 4 个阶段，由这 4 个阶段进行迭代。软件开发过程每迭代一次，软件开发又前进一个层次。采用螺旋模型的软件过程如图 3-8 所示。

需求定义
风险分析
工程实现
评审
1:N

图 3-8 螺旋模型的软件过程

螺旋模型基本做法是在“瀑布模型”的每一个开发阶段前引入一个非常严格的风险识别、风险分析和风险控制，它把软件项目分解成一个个小项目。每个小项目都标识一个或多个主要风险，直到所有的主要风险因素都被确定。

螺旋模型强调风险分析，使得开发人员和用户对每个演化层出现的风险有所了解，继而做出应有的反应，因此特别适用于庞大、复杂并具有高风险的系统。对于这些系统，风险是软件开发不可忽视且潜在的不利因素，它可能在不同程度上损害软件开发过程，影响软件产品的质量。减小软件风险的目标是在造成危害之

前，及时对风险进行识别及分析，决定采取何种对策，进而消除或减少风险的损害。

螺旋模型沿着螺线进行若干次迭代，图 3-9 中的 4 个象限代表了以下活动。

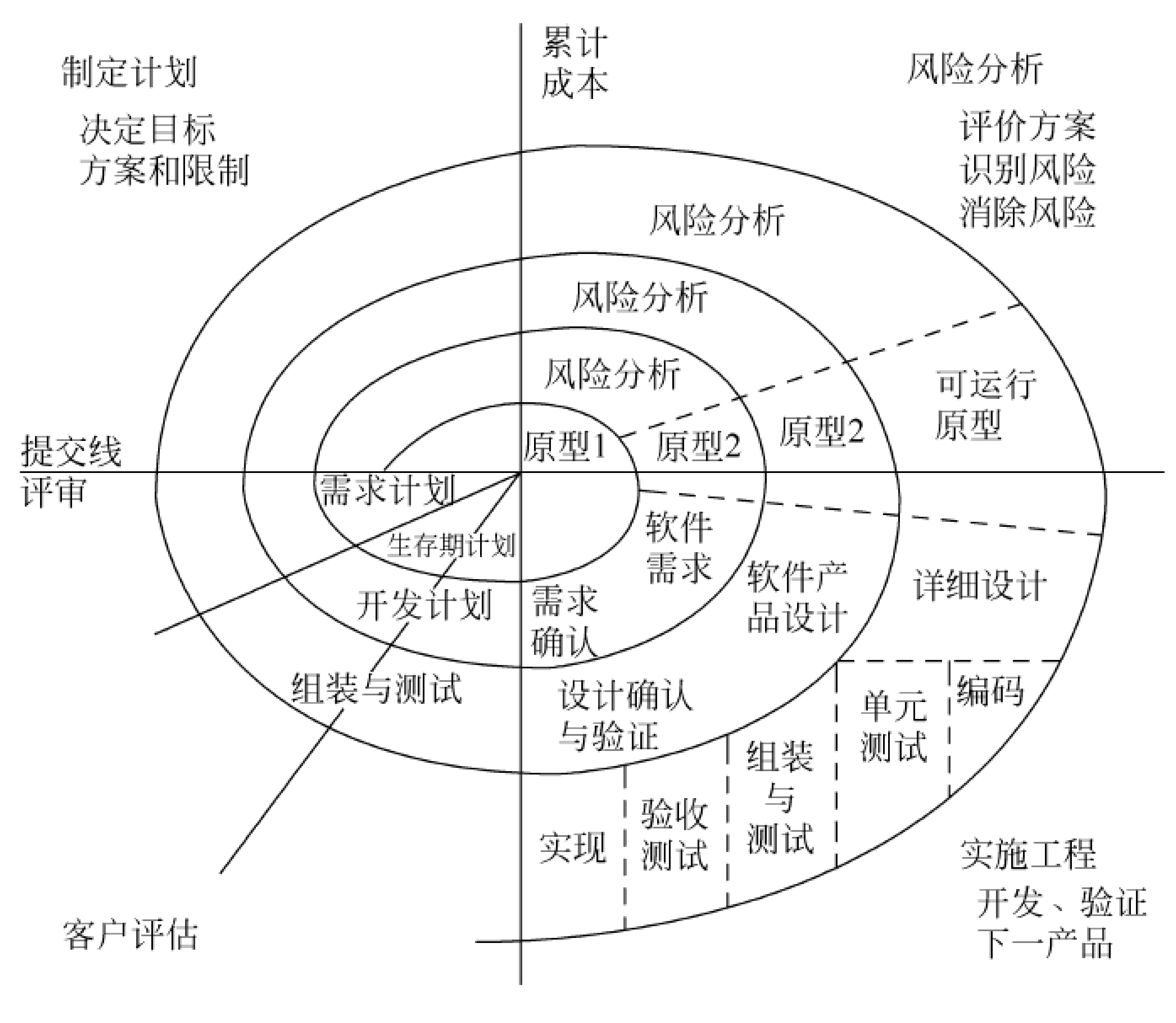


图 3-9 螺旋模型

(1) 制定计划：确定软件目标，选定实施方案，弄清项目开发的限制条件。

(2) 风险分析：分析评估所选方案，考虑如何识别和消除风险。

(3) 实施工程：实施软件开发和验证。

(4) 客户评估：评价开发工作，提出修正建议，制定下一步计划。

螺旋模型由风险驱动，强调可选方案和约束条件从而支持软件的重用，有助于将软件质量作为特殊目标融入产品开发之中。

螺旋模型很大程度上是一种风险驱动的方法体系，因为在每个阶段之前及经常发生的循环之前，都必须首先进行风险评估。在实践中，螺旋法技术和流程变得更为简单。迭代方法体系更倾向于按照开发/设计人员的方式工作，而不是项目经理的方式。螺旋模型中存在众多变量，并且在将来会有更大幅度的增长，该方法体系正良好运作着。表 3-1 是螺旋法能够解决的各种问题。

表 3-1 螺旋模型解决方案

经常遇到的问题	螺旋模型的解决方案
用户需求不够充分	允许并鼓励用户反馈信息
沟通不明	在项目早期就消除严重的曲解
刚性的体系	开发首先关注重要的业务和问题
主观臆断	通过测试和质量保证，做出客观的评估
潜在的不一致	在项目早期就发现不一致问题
糟糕的测试和质量保证	从第一次迭代就开始测试
采用瀑布法开发	在早期就找出并关注风险

（1）螺旋模型强调风险分析，但要求许多客户接受和相信这种分析，并做出相关反应是不容易的，因此，这种模型往往适应于内部的大规模软件开发。

（2）如果执行风险分析将大大影响项目的利润，那么进行风险分析毫无意义，因此，螺旋模型只适合大规模软件项目。

（3）软件开发人员应该擅长寻找可能的风险，准确地分析风险，否则将会带来更大的风险。

一个阶段首先是确定该阶段的目标，完成这些目标的选择方案及其约束条件，然后从风险角度分析方案的开发策略，努力排除各种潜在的风险，有时需要通过建造原型来完成。如果某些风险不能排除，该方案立即终止，否则启动下一个开发步骤。最后，评价该阶段的结果，并设计下一个阶段。

1）螺旋模型的优点

（1）设计上的灵活性，可以在项目的各个阶段进行变更。

（2）以小的分段来构建大型系统，使成本计算变得简单容易。

（3）客户始终参与每个阶段的开发，保证了项目不偏离正确方向以及项目的可控性。

（4）随着项目推进，客户始终掌握项目的最新信息，从而能够和管理层有效地交互。

（5）客户认可这种公司内部的开发方式带来的良好的沟通和高质量的产品。

2）螺旋模型的缺点

很难让用户确信这种演化方法的结果是可以控制的。建设周期长，而软件技术发展比较快，所以经常出现软件开发完毕后，和当前的技术水平有了较大的差距，无法满足当前用户需求。

对于新近开发，需求不明确的情况下，适合用螺旋模型进行开发，便于风险控制和需求变更。

3.3　单元测试

1. 单元测试概述

单元测试(Unit Testing)是指对软件中的最小可测试单元进行检查和验证。对于单元测试中单元的含义，一般来说，要根据实际情况去判定其具体含义，如C语言中单元指一个函数，Java里单元指一个类，图形化的软件中可以指一个窗口或一个菜单等。总地来说，单元就是人为规定的最小的被测功能模块。单元测试是在软件开发过程中要进行的最低级别的测试活动，软件的独立单元将在与程序的其他部分相隔离的情况下进行测试。

单元测试(模块测试)一般是开发者编写的一小段代码，用于检验被测代码的一个很小的、很明确的功能是否正确。通常而言，一个单元测试是用于判断某个特定条件(或者场景)下某个特定函数的行为。例如，可能把一个很大的值放入一个有序list中，然后确认该值出现在list的尾部。或者，可能会从字符串中删除匹配某种模式的字符，然后确认字符串中不再包含这些字符。

单元测试是由程序员自己来完成的，最终受益的也是程序员自己。可以这么说，程序员有责任编写功能代码，同时也就有责任为自己的代码编写单元测试。执行单元测试，就是为了证明这段代码的行为和期望功能一致。

工厂在组装一台电视机之前,会对每个元件都进行测试,这就是单元测试。其实我们每天都在做单元测试。编写一个函数,除了极简单的外,总是要执行一下,看看功能是否正常,有时还要想办法输出些数据,如弹出信息窗口,这也是单元测试。这种单元测试称为临时单元测试。只进行了临时单元测试的软件,针对代码的测试很不完整,代码覆盖率要超过70%都很困难,未覆盖的代码可能遗留大量的细小的错误,这些错误还会互相影响,当软件缺陷暴露出来的时候难于调试,大幅度提高后期测试和维护成本,也降低了开发商的竞争力。可以说,进行充分的单元测试,是提高软件质量,降低开发成本的必由之路。

对于程序员来说,如果养成了对自己写的代码进行单元测试的习惯,不但可以写出高质量的代码,而且还能提高编程水平。

要进行充分的单元测试,应专门编写测试代码,并与产品代码隔离。比较简单的办法是为产品工程建立对应的测试工程,为每个类建立对应的测试类,为每个函数(很简单的除外)建立测试函数。

一般认为,在结构化程序时代,单元测试所说的单元是指函数,在当今的面向对象时代,单元测试所说的单元是指类。以实践来看,以类作为测试单位,复杂度高,可操作性较差,因此仍然主张以函数作为单元测试的测试单位,但可以用一个测试类来组织某个类的所有测试函数。单元测试不应过分强调面向对象,因为局部代码依然是结构化的。单元测试的工作量较大,简单、实用、高效才是进行单元测试的硬道理。

有一种看法是,只测试类的接口(公有函数),不测试其他函数,从面向对象角度来看,确实有其道理,但是,测试的目的是找错并最终排错,因此,只要是包含错误的可能性较大的函数都要测试,跟函数是否私有没有关系。对于 C++语言来说,可以用一种简单的方法区隔需测试的函数:简单的函数如数据读写函数的实现在头文件中编写(inline 函数),所有在源文件中编写实现的函数都要进行测试(构造函数和析构函数除外)。

程序员编写代码时,一定会反复调试保证它能够编译通过。如果是编译没有通过的代码,没有任何人会愿意交付。但代码通过编译,只是说明了它的语法正确,无法保证它的语义也一定正确,没有任何人可以轻易承诺这段代码的行为一定是正确的。

幸运的是,单元测试会为单位功能的正确性承诺做保证。编写单元测试就是用来验证这段代码的行为是否与期望的一致。有了单元测试,程序员可以自信地交付自己的代码,而没有任何的后顾之忧。

单元测试与其他测试不同,单元测试可看作是编码工作的一部分,应该由程序员完成,也就是说,经过了单元测试的代码才是已完成的代码,提交产品代码时也要同时提交测试代码,测试部门可以做一定程度的审核。

2. 单元测试的任务

软件测试是为了发现错误而执行程序的过程。它是根据程序开发阶段的规格说明及程序内部结构而精心设计的一批测试用例(输入数据及其预期结果的集合),并利用该测试用例去运行程序,以发现错误的过程。软件测试的目的是为了尽早尽可能多地发现软件中的缺陷,提高软件产品的质量。

软件的单元测试是测试过程中最基本的测试,单元是软件的构成基础,因此单元的质量是整个软件质量的基础。单元测试是软件测试过程中直接与代码相关的测试,它对其他测

试步骤的进行有重要影响。单元测试在软件的编码阶段进行，主要完成的工作是根据详细设计说明书编写程序源代码，包括必要的数据文件，并进行单元测试，及时更正测试问题。

单元测试包括静态的代码审查和动态测试两个阶段。代码审查阶段对程序进行静态分析，包括编程风格检查、模块接口检查、程序语言检查、内存检查、比较和转移检查、性能检查、可维护性检查、逻辑检查、软件多余物检查等。动态测试阶段首先编写驱动模块和桩模块，在驱动模块和桩模块中设计相应的测试用例，测试用例应该覆盖单元模块的所有功能项，如果单元模块有性能等其他测试特性要求，则必须设计相应的测试用例测试这些特性，然后运行测试，比较测试结果。

单元测试的任务包括模块接口测试、模块局部数据结构测试、模块边界条件测试、模块中所有独立执行通路测试、模块的各条错误处理通路测试。模块测试是单元测试的基础。只有在数据能正确流入、流出模块的前提下，其他测试才有意义。检查局部数据结构是为了保证临时存储在模块内的数据在程序执行过程中完整、正确。除局部数据外，如果可能，单元测试时还应该查清全局数据对模块的影响。在模块中应对每一条独立执行路径进行测试，单元测试的基本任务是保证模块中每条语句至少执行一次。一个好的设计应能预见各种出错条件，并预设各种出错处理通路。边界条件测试是单元测试中最后也是最重要的一项任务，软件经常在边界上失效，采用边界值分析技术，针对边界值及其左右设计测试用例。

3. 单元测试工作内容及其流程

单元测试工作内容及其流程如表 3-2 和图 3-10 所示。

表 3-2　单元测试工作内容及其流程

活　　动	输　　入	输　　出	参与角色和职责
制定单元测试计划	1. 详细设计 2. 实现代码(可选)	单元测试计划(该计划可以不是一个独立的计划，可包含在实施计划中)	详细设计人员负责制定单元测试计划
设计单元测试	1. 单元测试计划 2. 详细设计 3. 实现代码(可选)	1. 单元测试用例 2. 已设计好的单元测试驱动块 3. 已设计好的单元测试桩模块	详细设计人员负责设计单元测试用例、设计驱动程序桩
实施单元测试	单元测试用例	1. 单元测试驱动模块 2. 单元测试桩模块	程序员负责编写测试驱动程序和稳定桩
执行单元测试	1. 实现代码 2. 单元测试计划 3. 单元测试用例 4. 被测试单元 5. 单元测试驱动模块和桩模块	1. 测试结果 2. 测试问题报告	程序员执行测试并记录测试结果
评估单元测试	1. 单元测试计划 2. 测试结果	测试评估报告	详细设计人员负责评估此次测试并生成测试评估报告

4. 单元测试用例设计方法

现在的软件几乎都是用事件触发来控制流程的，事件触发时的情景便形成了场景，而同一事件不同的触发顺序和处理结果就形成事件流。这种在软件设计方面的思想也可以引入

到软件测试中，可以比较生动地描绘出事件触发时的情景，有利于测试设计者设计测试用例，同时使测试用例更容易理解和执行。

基本流和备选流：如图 3-11 所示，图中经过用例的每条路径都用基本流和备选流来表示，直黑线表示基本流，是经过用例的最简单的路径。备选流可以用不同的色彩和样式表示，一个备选流可能从基本流开始，在某个特定条件下执行，然后重新加入基本流中（如备选流 1 和 3）；也可能起源于另一个备选流（如备选流 2），或者终止用例而不再重新加入到某个流（如备选流 2 和 4）。

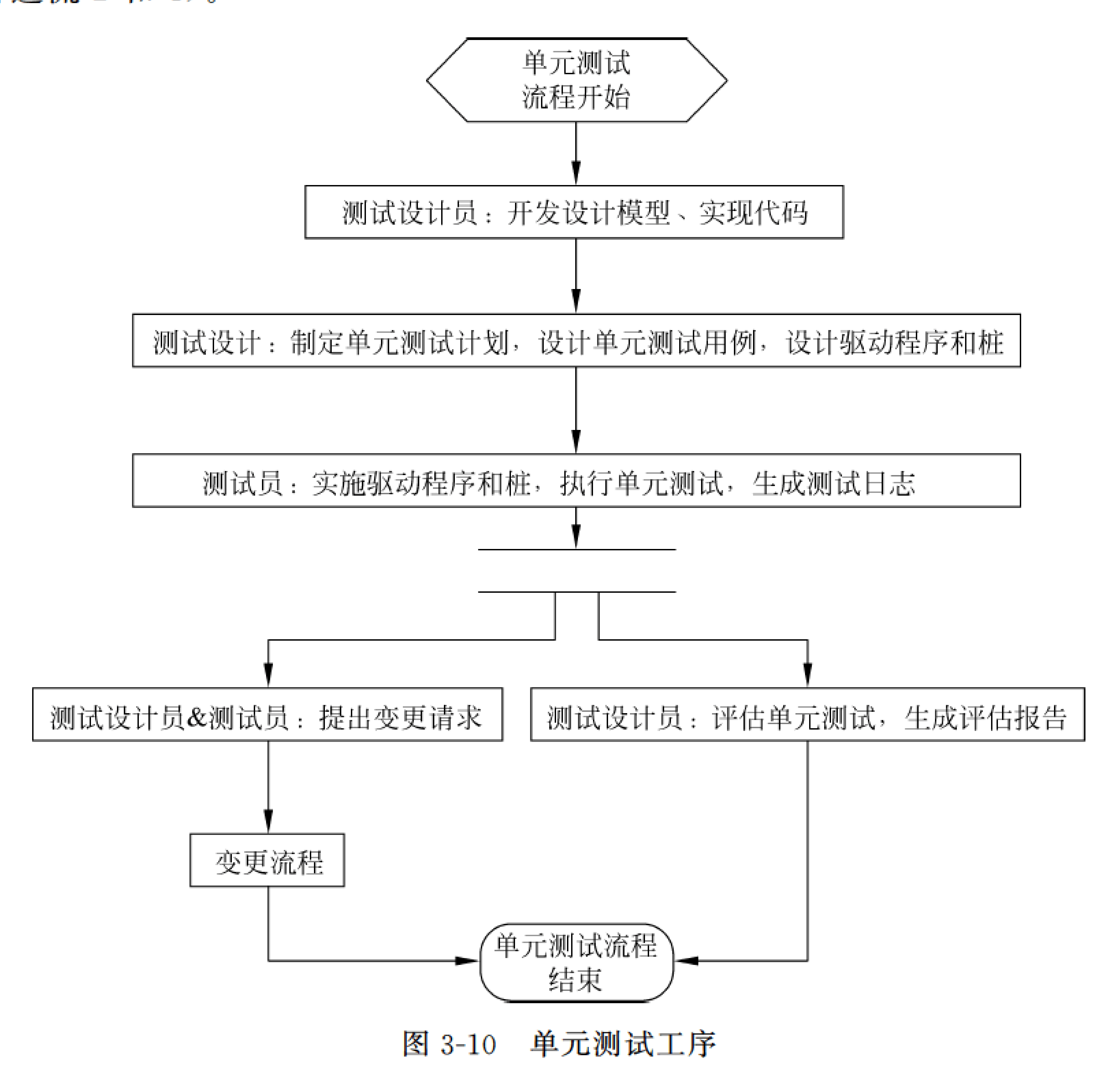


图 3-10　单元测试工序

案例：ATM 的流程示意图如图 3-12 所示。

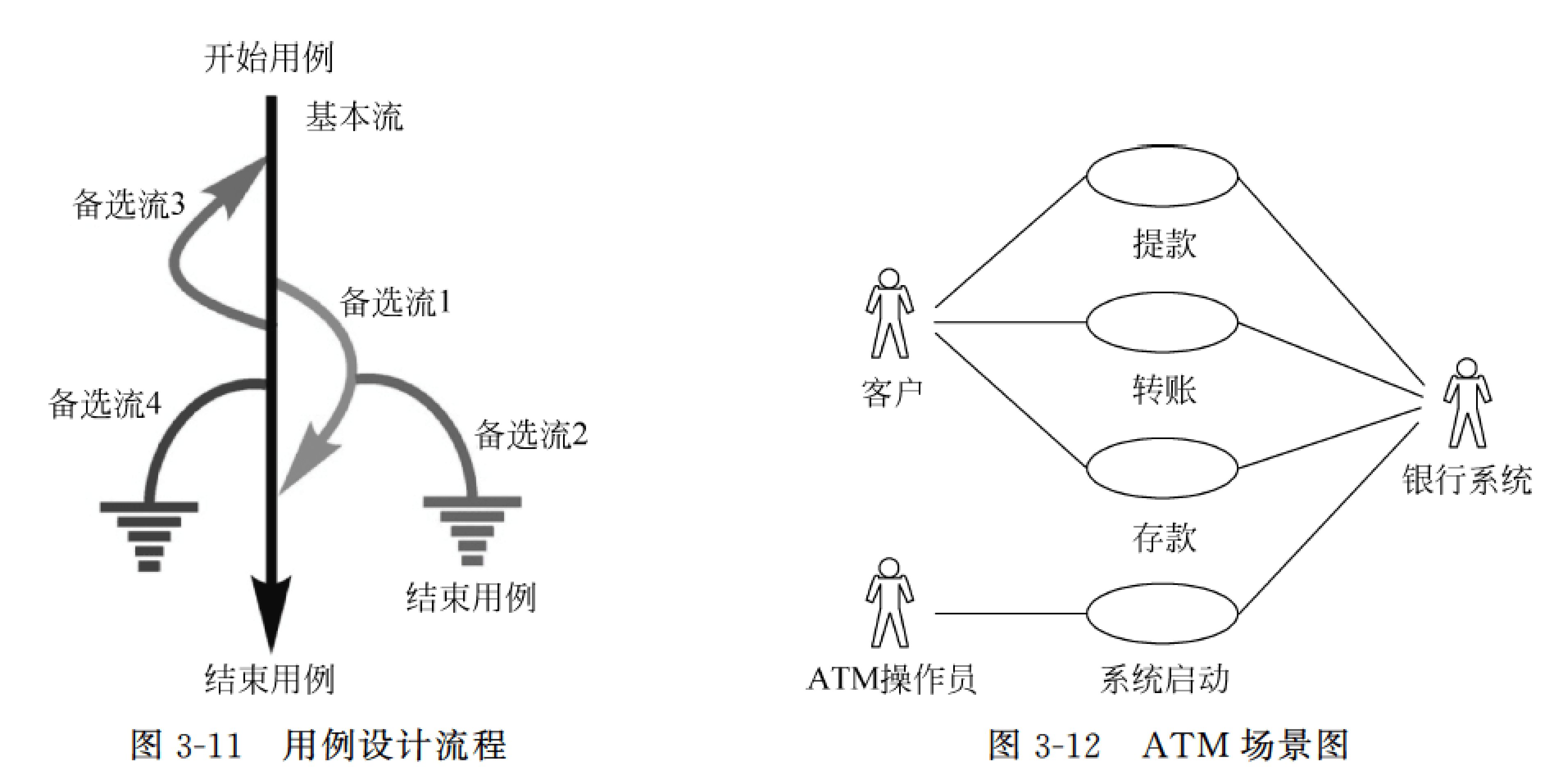


图 3-11　用例设计流程

图 3-12　ATM 场景图

场景设计：如表 3-3 所示是生成的 ATM 场景。

表 3-3　场景设计

场景 1——成功提款	基本流	
场景 2——ATM 内没有现金	基本流	备选流 2
场景 3——ATM 内现金不足	基本流	备选流 3
场景 4——PIN 有误(还有输入机会)	基本流	备选流 4
场景 5——PIN 有误(不再有输入机会)	基本流	备选流 4
场景 6——账户不存在/账户类型有误	基本流	备选流 5
场景 7——账户余额不足	基本流	备选流 6

注：为方便起见，备选流 3 和 6(场景 3 和 7)内的循环以及循环组合未纳入表 3-3 中。

用例设计：对于这 7 个场景中的每一个场景都需要确定测试用例。可以采用矩阵或决策表来确定和管理测试用例。下面显示了一种通用格式，其中各行代表各个测试用例，而各列则代表测试用例的信息。本示例中，对于每个测试用例，存在一个测试用例 ID、条件(或说明)、测试用例中涉及的所有数据元素(作为输入或已经存在于数据库中)以及预期结果。

表 3-4　测试用例

TC(测试用例)ID 号	场景/条件	PIN	账号	输入(或选择)的金额	账面金额	ATM 内的金额	预期结果
CW1	场景 1：成功提款	V	V	V	V	V	成功提款
CW2	场景 2：ATM 内没有现金	V	V	V	V	I	提款选项不可用，用例结束
CW3	场景 3：ATM 内现金不足	V	V	V	V	I	警告消息，返回基本流步骤 6，输入金额
CW4	场景 4：PIN 有误(还有不止一次输入机会)	I	V	n/a	V	V	警告消息，返回基本流步骤 4，输入 PIN
CW5	场景 4：PIN 有误(还有一次输入机会)	I	V	n/a	V	V	警告消息，返回基本流步骤 4，输入 PIN
CW6	场景 4：PIN 有误(不再有输入机会)	I	V	n/a	V	V	警告消息，卡予以保留，用例结束

数据设计：一旦确定了所有的测试用例，则应对这些用例进行复审和验证以确保其准确且适度，并取消多余或等效的测试用例。

测试用例一经认可，就可以确定实际数据值(在测试用例实施矩阵中)并且设定测试数据，如表 3-5 所示。

表 3-5　测试用例表

TC(测试用例)ID 号	场景/条件	PIN	账号	输入(或选择)的金额/元	账面金额/元	ATM 内的金额/元	预期结果
CW1	场景 1：成功提款	4987	809-498	50.00	500.0	2000	成功提款。账户余额被更新为 450.00
CW2	场景 2：ATM 内没有现金	4987	809-498	100.00	500.0	0.00	提款选项不可用，用例结束

续表

TC(测试用例)ID 号	场景/条件	PIN	账号	输入(或选择)的金额/元	账面金额/元	ATM 内的金额/元	预 期 结 果
CW3	场景 3：ATM 内现金不足	4987	809-498	100.00	500.0	70.00	警告消息，返回基本流步骤 6，输入金额
CW4	场景 4：PIN 有误（还有不止一次输入机会）	4978	809-498	n/a	500.00	2000	警告消息，返回基本流步骤 4，输入 PIN
CW5	场景 4：PIN 有误（还有一次输入机会）	4978	809-498	n/a	500.00	2000	警告消息，返回基本流步骤 4，输入 PIN
CW6	场景 4：PIN 有误（不再有输入机会）	4978	809-498	n/a	500.00	2000	警告消息，卡予以保留，用例结束

1）等价类划分法

等价类是指某个输入域的子集合。在该子集合中，各个输入数据对于揭露程序中的错误都是等效的，并合理地假定：测试某等价类的代表值就等于对这一类其他值的测试，因此，可以把全部输入数据合理划分为若干等价类，在每一个等价类中取一个数据作为测试的输入条件就可以用少量代表性的测试数据取得较好的测试结果。等价类划分可有两种不同的情况：有效等价类和无效等价类。

（1）有效等价类

它是指对于程序的规格说明来说是合理的、有意义的输入数据构成的集合。利用有效等价类可检验程序是否实现了规格说明中所规定的功能和性能。

（2）无效等价类

它与有效等价类的定义恰巧相反。无效等价类指对程序的规格说明是不合理的或无意义的输入数据所构成的集合。对于具体的问题，无效等价类至少应有一个，也可能有多个。

设计测试用例时，要同时考虑这两种等价类。因为软件不仅要能接收合理的数据，也要能经受意外的考验，这样的测试才能确保软件具有更高的可靠性。

（3）划分等价类的标准

① 完备测试、避免冗余。

② 划分等价类重要的是：集合的划分，划分为互不相交的一组子集，而子集的并是整个集合。

③ 并是整个集合：完备性。

④ 子集互不相交：保证一种形式的无冗余性。

⑤ 同一类中标识（选择）一个测试用例，同一等价类中，往往处理相同，相同处理映射到“相同的执行路径”。

（4）划分等价类的方法

① 在输入条件规定了取值范围或值的个数的情况下，则可以确立一个有效等价类和两个无效等价类。例如，输入值是学生成绩，范围是 0～100，如图 3-13 所示。

② 在输入条件规定了输入值的集合或者规定了“必须如何”的条件的情况下，可确立一个有效等价类和一个无效等价类。

③ 在输入条件是一个布尔量的情况下，可确定一个有效等价类和一个无效等价类。

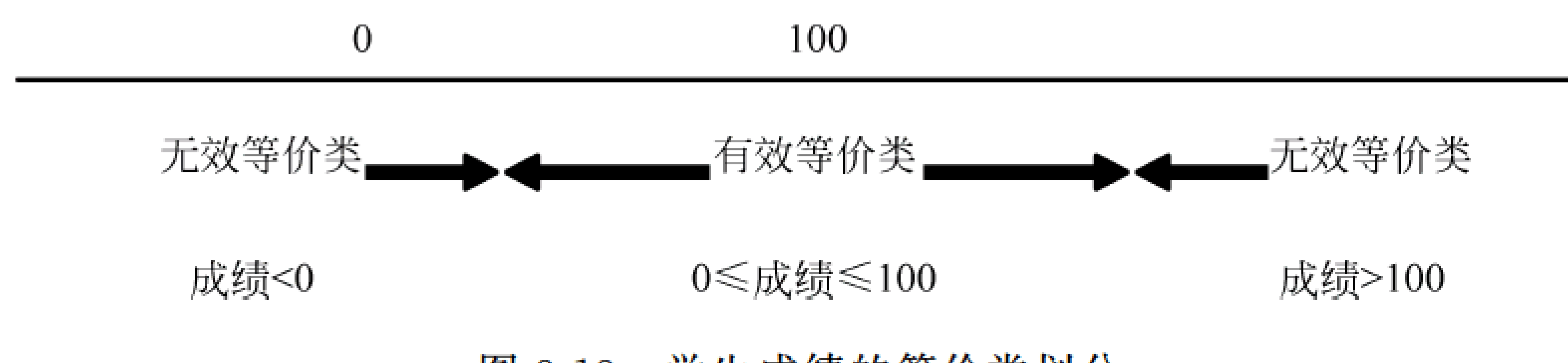


图 3-13 学生成绩的等价类划分

④ 在规定了输入数据的一组值(假定 n 个),并且程序要对每一个输入值分别处理的情况下,可确立 n 个有效等价类和一个无效等价类。

例如,输入条件说明学历可为专科、本科、硕士、博士四种之一,则分别取 4 个值作为 4 个有效等价类,另外把 4 种学历之外的任何学历作为无效等价类。

⑤ 在规定输入数据必须遵守的规则的情况下,可确立一个有效等价类(符合规则)和若干个无效等价类(从不同角度违反规则)。

⑥ 在确知已划分的等价类中各元素在程序处理中的方式不同的情况下,则应再将该等价类进一步地划分为更小的等价类。

(5) 设计测试用例

在确立等价类后,可建立等价类表,列出所有划分出的等价类输入条件。然后从划分出的等价类中按以下三个原则设计测试用例。

① 为每一个等价类规定一个唯一的编号;

② 设计一个新的测试用例,使其尽可能多地覆盖尚未被覆盖的有效等价类,重复这一步,直到所有的有效等价类都被覆盖为止;

③ 设计一个新的测试用例,使其仅覆盖一个尚未被覆盖的无效等价类,重复这一步,直到所有的无效等价类都被覆盖为止。

2) 边界值分析方法

边界值分析法就是对输入或输出的边界值进行测试的一种黑盒测试方法。通常边界值分析法是作为对等价类划分法的补充,在这种情况下,其测试用例来自等价类的边界。

长期的测试工作经验告诉我们,大量的错误是发生在输入或输出范围的边界上,而不是发生在输入输出范围的内部。因此针对各种边界情况设计测试用例,可以查出更多的错误。

使用边界值分析方法设计测试用例,首先应确定边界情况。通常输入和输出等价类的边界,就是应着重测试的边界情况。应当选取正好等于,刚刚大于或刚刚小于边界的值作为测试数据,而不是选取等价类中的典型值或任意值作为测试数据。

常见的边界值有以下几种。

① 对 16b 的整数而言,32 767 和—32 768 是边界。

② 屏幕上光标在最左上、最右下位置。

③ 报表的第一行和最后一行。

④ 数组元素的第一个和最后一个。

⑤ 循环的第 0 次、第一次和倒数第二次、最后一次。

边界值分析使用与等价类划分法相同的划分,只是边界值分析假定错误更多地存在于

划分的边界上，因此在等价类的边界上以及两侧的情况设计测试用例。

例如，测试计算平方根的函数。

输入：实数

输出：实数

规格说明：当输入一个0或比0大的数的时候，返回其正平方根；当输入一个小于0的数时，显示错误信息“平方根非法-输入值小于0”并返回0；库函数Print-Line可以用来输出错误信息。

(1) 可以考虑做出如下划分。

输入：①<0和②≥0。

输出：①≥0和②Error。

(2) 测试用例有两个：

① 输入4，输出2。

② 输入−10，输出0和错误提示。

第一组划分的边界为0和最大正实数；第二组划分的边界为最小负实数和0。由此得到以下测试用例。

(1) 输入{最小负实数}

(2) 输入{绝对值很小的负数}

(3) 输入0

(4) 输入{绝对值很小的正数}

(5) 输入{最大正实数}

常见等价类划分方法如表3-6所示。

表3-6 常见等价类划分方法

项	边界值	测试用例的设计思路
字符	起始−1个字符/结束+1个字符	假设一个文本输入区域允许输入1～255个字符，输入1个和255个字符作为有效等价类；输入0个和256个字符作为无效等价类，这几个数值都属于边界条件值
数值	最小值−1/最大值+1	假设某软件的数据输入域要求输入5位的数据值，可以使用10 000作为最小值、99 999作为最大值；然后使用刚好小于5位和大于5位的数值来作为边界条件
空间	小于空余空间一点儿/大于满空间一点儿	例如，在用U盘存储数据时，使用比剩余磁盘空间大一点儿(几KB)的文件作为边界条件

在多数情况下，边界值条件是基于应用程序的功能设计而需要考虑的因素，可以从软件的规格说明或常识中得到，也是最终用户可以很容易发现问题的。然而，在测试用例设计过程中，某些边界值条件是不需要呈现给用户的，或者说用户是很难注意到的，但同时确实属于检验范畴内的边界条件，称为内部边界值条件或子边界值条件。

内部边界值条件主要有下面几种。

(1) 数值的边界值检验：计算机是基于二进制进行工作的，因此，软件的任何数值运算都有一定的范围限制，如表3-7所示。

表 3-7　数值的边界值

项	范围或值
位(bit)	0 或 1
字节(byte)	0～255
字(word)	0～65 535(单字)或 0～4 294 967 295(双字)
千(K)	1024
兆(M)	1 048 576
吉(G)	1 073 741 824

(2) 字符的边界值检验：在计算机软件中，字符也是很重要的表示元素，其中 ASCII 和 Unicode 是常见的编码方式。表 3-8 中列出了一些常用字符对应的 ASCII 码值。

表 3-8　字符的边界值

字符	ASCII 码值	字符	ASCII 码值
空(null)	0	A	65
空格(space)	32	a	97
斜杠(/)	47	Z	90
0	48	z	122
冒号(：)	58	单引号(‘)	96
@	64		

基于边界值分析方法选择测试用例的原则如下。

(1) 如果输入条件规定了值的范围，则应取刚达到这个范围的边界的值，以及刚刚超越这个范围边界的值作为测试输入数据。

例如，如果程序的规格说明中规定："重量在 10～50kg 范围内的邮件，其邮费计算公式为……"作为测试用例，应取 10 及 50，还应取 10.01，49.99，9.99 及 50.01 等。

(2) 如果输入条件规定了值的个数，则用最大个数，最小个数，比最小个数少 1，比最大个数多 1 的数作为测试数据。

例如，一个输入文件应包括 1～255 个记录，则测试用例可取 1 和 255，还应取 0 及 256 等。

(3) 将规则(1)和(2)应用于输出条件，即设计测试用例使输出值达到边界值及其左右的值。

例如，某程序的规格说明要求计算出"每月保险金扣除额为 0～1165.25 元"，其测试用例可取 0.00 及 1165.24、还可取 0.01 及 1165.26 等。

再如一程序属于情报检索系统，要求每次"最少显示 1 条、最多显示 4 条情报摘要"，这时应考虑的测试用例包括 1 和 4，还应包括 0 和 5 等。

(4) 如果程序的规格说明给出的输入域或输出域是有序集合，则应选取集合的第一个元素和最后一个元素作为测试用例。

(5) 如果程序中使用了一个内部数据结构，则应当选择这个内部数据结构的边界上的值作为测试用例。

(6) 分析规格说明，找出其他可能的边界条件。

案例：现有一个学生标准化考试批阅试卷，产生成绩报告的程序。其规格说明如下：

程序的输入文件由一些有 80 个字符的记录组成，如图 3-14 所示，所有记录分为三组。

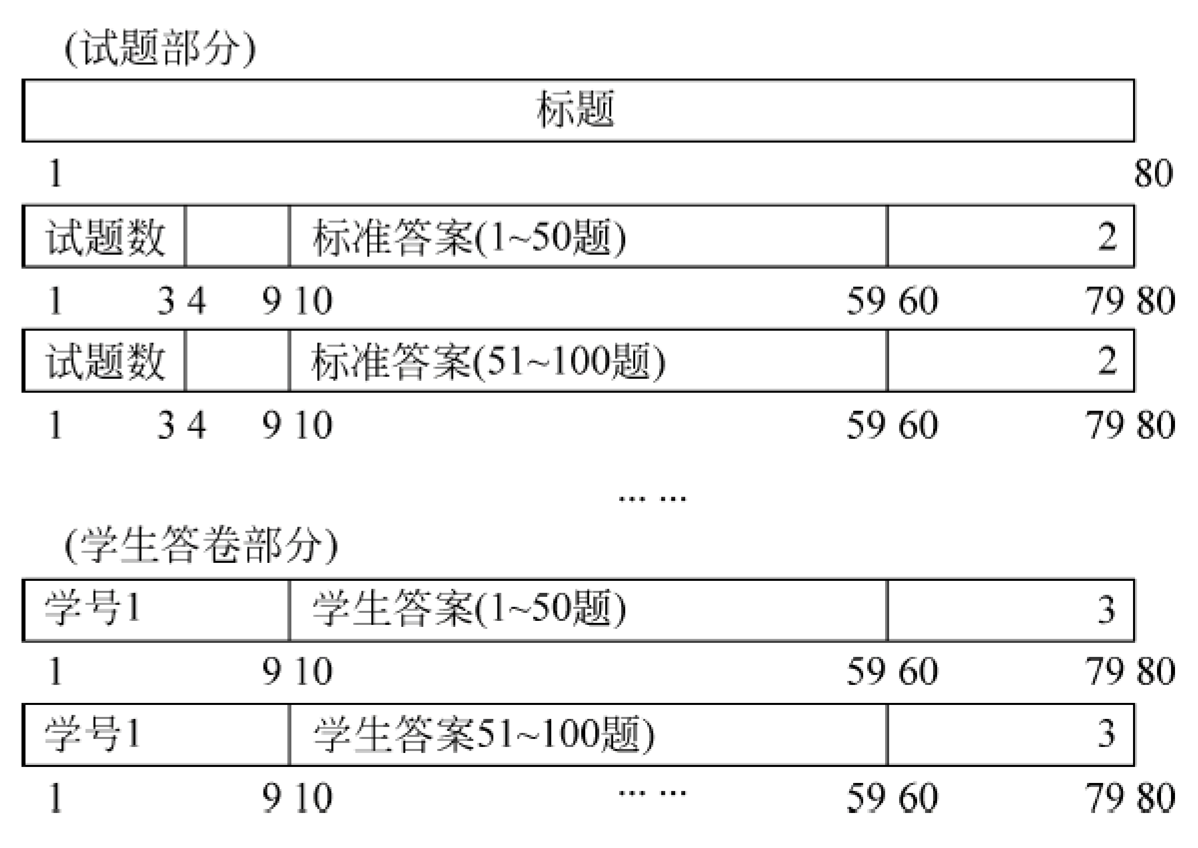


图 3-14 试卷批阅输入文件记录

(1) 标题：这一组只有一个记录，其内容为输出成绩报告的名字。

(2) 试卷各题标准答案记录：每个记录均在第 80 个字符处标以数字“2”。该组的第一个记录的第 1～3 个字符为题目编号(取值为 1～999)。第 10～59 个字符给出第 1～50 题的答案(每个合法字符表示一个答案)。该组的第 2，第 3…个记录相应为第 51～100，第 101～150，…题的答案。

输入条件及相应测试用例如表 3-9 所示。

表 3-9 输入条件及相应测试用例

输入条件	测试用例
输入文件	空输入文件
标题	没有标题
	标题只有一个字符
	标题有 80 个字符
试题数	试题数为 1
	试题数为 50
	试题数为 51
	试题数为 100
	试题数为 0
	试题数含有非数字字符
标准答案记录	没有标准答案记录，有标题
	标准答案记录多于一个
	标准答案记录少一个
学生人数	0 个学生
	1 个学生
	200 个学生
	201 个学生
学生答题	某学生只有一个回答记录，但有两个标准答案记录
	该学生是文件中的第一个学生
	该学生是文件中的最后一个学生(记录数出错的学生)

续表

输入条件	测 试 用 例
学生答题	某学生有两个回答记录，但只有一个标准答案记录
	该学生是文件中的第一个学生（记录数出错的学生）
	该学生是文件中的最后一个学生
学生成绩	所有学生的成绩都相等
	每个学生的成绩都不相等
	部分学生的成绩相同
	（检查是否能按成绩正确排名次）
	有个学生 0 分
	有个学生 100 分

表 3-10 是输出条件及相应的测试用例表。

表 3-10　输出条件及相应的测试用例表

输出条件	测 试 用 例
输出报告 a、b	有个学生的学号最小（检查按序号排序是否正确）
	有个学生的号最大（检查按序号排序是否正确）
	适当的学生人数，使产生的报告刚好满一页（检查打印页数）
	学生人数比刚才多出 1 人（检查打印换页）
输出报告 c	平均成绩 100
	平均成绩 0
	标准偏差为最大值（有一半的 0 分，其他 100 分）
	标准偏差为 0（所有成绩相等）
输出报告 d	所有学生都答对了第一题
	所有学生都答错了第一题
	所有学生都答对了最后一题
	所有学生都答错了最后一题
	选择适当的试题数，是第 4 个报告刚好打满一页
	试题数比刚才多 1，使报告打满一页后，刚好剩下一题未打

（3）每个学生的答卷描述：该组中每个记录的第 80 个字符均为数字“3”。每个学生的答卷在若干个记录中给出。如甲的首记录第 1～9 字符给出学生姓名及学号，第 10～59 字符列出的是甲所做的第 1～50 题的答案。若试题数超过 50，则第 2，第 3…记录分别给出他的第 51～100，第 101～150…题的解答。然后是学生乙的答卷记录。

（4）学生人数不超过 200，试题数不超过 999。

（5）程序的输出有 4 个报告：

① 按学号排列的成绩单，列出每个学生的成绩、名次。

② 按学生成绩排序的成绩单。

③ 平均分数及标准偏差的报告。

④ 试题分析报告。按试题号排序，列出各题学生答对的百分比。

解答：分别考虑输入条件和输出条件，以及边界条件。给出如表 3-9 所示的输入条件及相应的测试用例。

3）因果图法

因果图法是一种利用图解法分析输入的各种组合情况，从而设计测试用例的方法，它适合检查程序输入条件的各种组合情况。

等价类划分法和边界值分析方法都是着重考虑输入条件，但没有考虑输入条件的各种组合、输入条件之间的相互制约关系。这样虽然各种输入条件可能出错的情况已经测试到，但多个输入条件组合起来可能出错的情况却被忽视了。

如果在测试时必须考虑输入条件的各种组合，则可能的组合数目将是天文数字，因此必须考虑采用一种适合描述多种条件的组合、相应产生多个动作的形式来进行测试用例的设计，这就需要利用因果图(逻辑模型)。

图 3-15 中 4 种符号分别表示了规格说明中的 4 种因果关系。

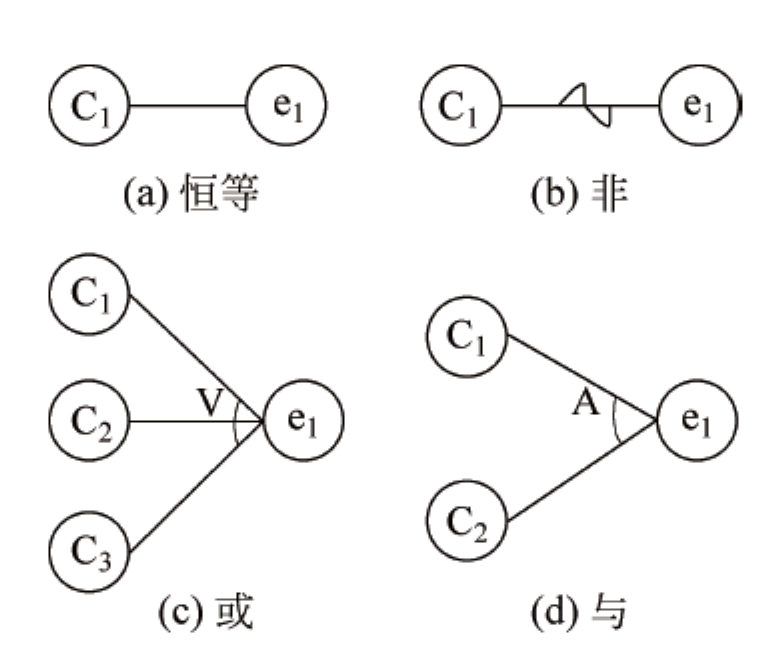


图 3-15 4 种因果关系

因果图中使用了简单的逻辑符号，以直线连接左右节点。左节点表示输入状态(或称原因)，右节点表示输出状态(或称结果)。C_i 表示原因，通常置于图的左部；e_i 表示结果，通常在图的右部。C_i 和 e_i 均可取值 0 或 1，0 表示某状态不出现，1 表示某状态出现。

(1) 关系

① 恒等：若 C_i 是 1，则 e_i 也是 1；否则 e_i 为 0。

② 非：若 C_i 是 1，则 e_i 是 0；否则 e_i 是 1。

③ 或：若 C_1 或 C_2 或 C_3 是 1，则 e_i 是 1；否则 e_i 为 0。"或"可有任意个输入。

④ 与：若 C_1 和 C_2 都是 1，则 e_i 为 1；否则 e_i 为 0。"与"也可有任意个输入。

(2) 约束

输入状态相互之间还可能存在某些依赖关系，称为约束。例如，某些输入条件本身不可能同时出现。输出状态之间也往往存在约束。在因果图中，用特定的符号标明这些约束。

① 输入条件约束类型

- E 约束(异)：a 和 b 中至多有一个可能为 1，即 a 和 b 不能同时为 1。
- I 约束(或)：a、b 和 c 中至少有一个必须是 1，即 a、b 和 c 不能同时为 0。
- O 约束(唯一)：a 和 b 必须有一个，且仅有一个为 1。
- R 约束(要求)：a 是 1 时，b 必须是 1，即不可能 a 是 1 时 b 是 0。

② 输出条件约束类型

输出条件的约束只有 M 约束(强制)：若结果 a 是 1，则结果 b 强制为 0。

因果图输入、输出条件约束如图 3-16 所示。采用因果图法设计测试用例的步骤如下。

(1) 分析软件规格说明描述中，哪些是原因(即输入条件或输入条件的等价类)，哪些是结果(即输出条件)，并给每个原因和结果赋予一个标识符。

(2) 分析软件规格说明描述中的语义，找出原因与结果之间、原因与原因之间对应的关系，根据这些关系画出因果图。

(3) 由于语法或环境限制，有些原因与原因之间，原因与结果之间的组合情况不可能出

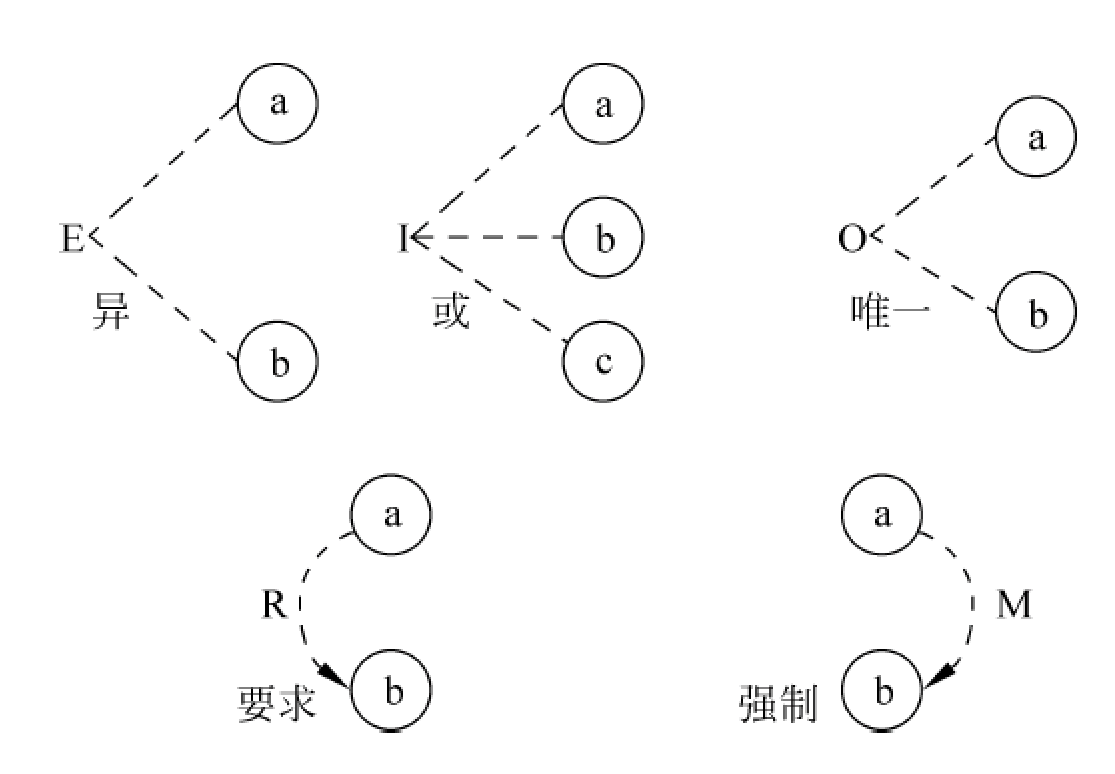


图 3-16　因果图约束

现，为表明这些特殊情况，在因果图上用一些记号表明约束或限制条件。

（4）把因果图转换为判定表。

（5）把判定表的每一列拿出来作为依据，设计测试用例。

案例：某软件规格说明书包含这样的要求，第一列字符必须是 A 或 B，第二列字符必须是一个数字，在此情况下进行文件的修改，但如果第一列字符不正确，则给出信息 L；如果第二列字符不是数字，则给出信息 M。

解答：

（1）根据题意，原因和结果如下：

原因：

1——第一列字符是 A；

2——第一列字符是 B；

3——第二列字符是一数字。

结果：

21——修改文件；

22 ——给出信息 L；

23——给出信息 M。

（2）其对应的因果图如图 3-17 所示。

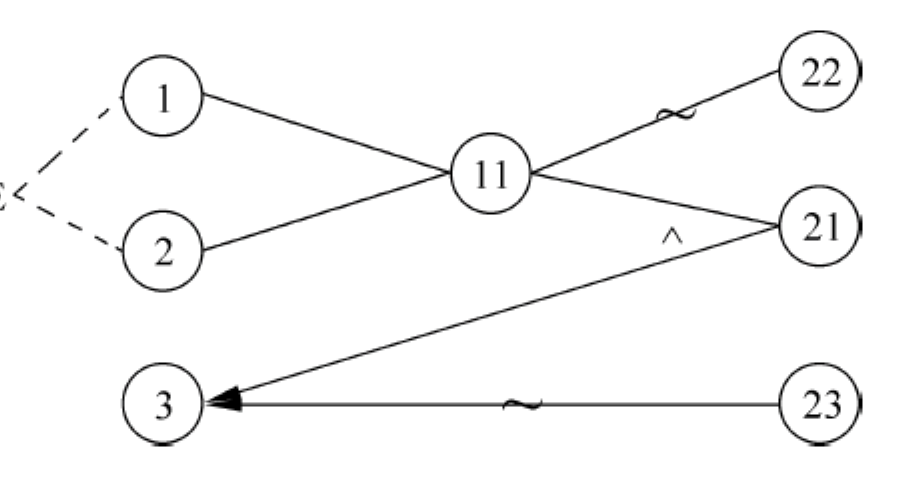


图 3-17　因果图

11 为中间节点；考虑到原因 1 和原因 2 不可能同时为 1，故在因果图上施加 E 约束。

（3）根据因果图建立判定表，如表 3-11 所示。

表 3-11　判定表

		1	2	3	4	5	6	7	8
条件（原因）	1	1	1	1	1	0	0	0	0
	2	1	1	0	0	1	1	0	0
	3	1	0	1	0	1	0	1	0
	11			1	1	1	1	0	0

续表

		1	2	3	4	5	6	7	8
动作(结果)	22			0	0	0	0	1	1
	21			1	0	1	0	0	0
	23			0	1	0	1	0	1
测试用例				A3	AM	B5	BN	C2	DY
				A8	A?	B4	B!	X6	P;

表 3-11 中 8 种情况的左面两列情况中,原因①和原因②同时为 1,这是不可能出现的,故应排除这两种情况。表的最下一栏给出了 6 种情况的测试用例,这是我们所需要的数据。

4) 错误推测方法

错误推测方法是基于经验和直觉推测程序中所有可能存在的各种错误,从而有针对性地设计测试用例的方法。

列举出程序中所有可能有的错误和容易发生错误的特殊情况,根据它们选择测试用例。

例如,输入数据和输出数据为 0 的情况;输入表格为空格或输入表格只有一行。这些都是容易发生错误的情况。可选择这些情况下的例子作为测试用例。

再如,前面例子中成绩报告的程序,采用错误推测法还可补充设计以下一些测试用例。

(1) 程序是否把空格作为回答。

(2) 在回答记录中混有标准答案记录。

(3) 除了标题记录外,还有一些记录最后一个字符既不是 2 也不是 3。

(4) 有两个学生的学号相同。

(5) 试题数是负数。

再如,测试一个对线性表(例如数组)进行排序的程序,可推测列出以下几项需要特别测试的情况。

(1) 输入的线性表为空表;

(2) 表中只含有一个元素;

(3) 输入表中所有元素已排好序;

(4) 输入表已按逆序排好;

(5) 输入表中部分或全部元素相同。

3.4 集成测试

集成测试(也叫组装测试,联合测试)是单元测试的逻辑扩展。它的最简单的形式是:两个已经测试过的单元组合成一个组件,并且测试它们之间的接口。从这一层意义上讲,组件是指多个单元的集成聚合。在现实方案中,许多单元组合成组件,而这些组件又聚合成程序的更大部分。方法是测试片段的组合,并最终扩展进程,将用户的模块与其他组的模块一起测试。最后,将构成进程的所有模块一起测试。此外,如果程序由多个进程组成,应该成对测试它们,而不是同时测试所有进程。

集成测试识别组合单元时出现的问题。通过使用要求在组合单元前测试每个单元并确保每个单元的生存能力的测试计划，可以知道在组合单元时所发现的任何错误很可能与单元之间的接口有关。这种方法将可能发生的情况数量减少到更简单的分析级别。

集成测试是在单元测试的基础上，测试在将所有的软件单元按照概要设计规格说明的要求组装成模块、子系统或系统的过程中各部分工作是否达到或实现相应技术指标及要求的活动。也就是说，在集成测试之前，单元测试应该已经完成，集成测试中所使用的对象应该是已经经过单元测试的软件单元。这一点很重要，因为如果不经过单元测试，那么集成测试的效果将会受到很大影响，并且会大幅增加软件单元代码纠错的代价。

集成测试是单元测试的逻辑扩展。在现实方案中，集成是指多个单元的聚合，许多单元组合成模块，而这些模块又聚合成程序的更大部分，如分系统或系统。集成测试采用的方法是测试软件单元的组合能否正常工作，以及与其他组的模块能否集成起来工作。最后，还要测试构成系统的所有模块组合能否正常工作。

所有的软件项目都不能摆脱系统集成这个阶段。不管采用什么开发模式，具体的开发工作总要从一个一个的软件单元做起，软件单元只有经过集成才能形成一个有机的整体。具体的集成过程可能是显性的也可能是隐性的。只要有集成，总是会出现一些常见问题，工程实践中，几乎不存在软件单元组装过程中不出任何问题的情况。

集成测试的必要性还在于一些模块虽然能够单独地工作，但并不能保证连接起来也能正常工作。程序在某些局部反映不出来的问题，有可能在全局上会暴露出来，影响功能的实现。此外，在某些开发模式中，如迭代式开发，设计和实现是迭代进行的。在这种情况下，集成测试的意义还在于它能间接地验证概要设计是否具有可行性。

集成测试的目的是确保各单元组合在一起后能够按既定意图协作运行，并确保增量的行为正确。它所测试的内容包括单元间的接口以及集成后的功能。使用黑盒测试方法测试集成的功能，并且对以前的集成进行回归测试。

1. 集成测试过程

1）软件测试过程图解

软件测试过程如图 3-18 所示。

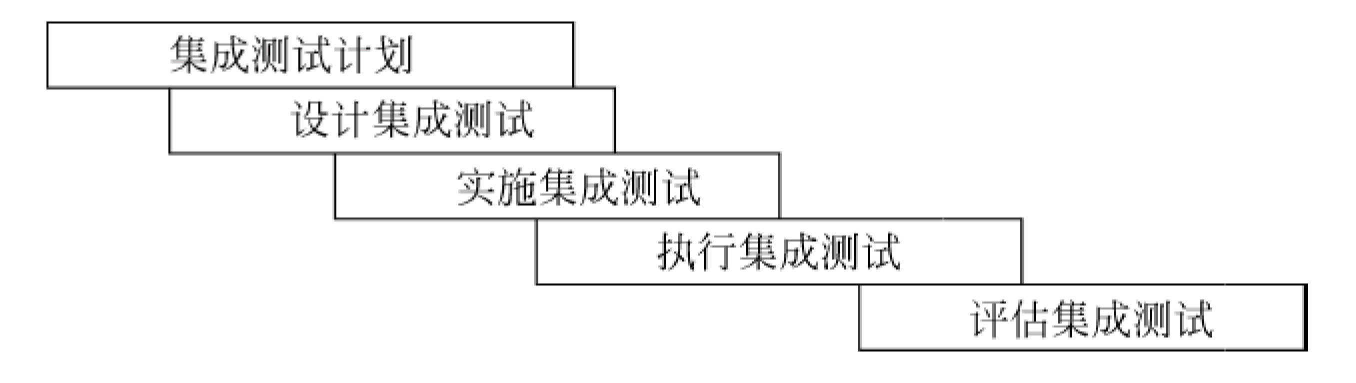


图 3-18　软件测试过程

2）集成测试需求获取

集成测试需求所确定的是对某一集成工作版本的测试的内容，即测试的具体对象。集成测试需求主要来源于设计模型(Design Model)和集成构件计划(Integration Build Plan)。集成测试着重于集成版本的外部接口的行为。因此，测试需求需具有可观测、可测评性。

(1) 集成工作版本应分析其类协作与消息队列，从而找出该工作版本的外部接口。

（2）由集成工作版本的外部接口确定集成测试用例。

（3）测试用例应覆盖工作版本每一外部接口的所有消息流序列。

注意：一个外部接口和测试用例的关系是多对多，部分集成工作版本的测试需求可映射到系统测试需求，因此对这些集成测试用例可采用重用系统测试用例技术。

3）集成测试工作机制

软件集成测试工作需要产品评测部担任，需要项目组相关角色配合完成。具体角色和职责如表 3-12 和表 3-13 所示。

表 3-12　评测部的角色和职责

角色	职　　责
测试设计员	负责制定集成测试计划、设计集成测试、实施集成测试、评估集成测试
测试员	执行集成测试，记录测试结果

表 3-13　软件项目组的角色和职责

角色	职　　责
实施员	负责实施类(包括驱动程序和桩)，并对其进行单元测试。根据集成测试发现的缺陷提出变更申请
配置管理员	负责对测试工件进行配置管理
集成员	负责制定集成构建计划，按照集成计划将通过了单元测试的类集成
设计员	负责设计测试驱动程序和桩。根据集成测试发现的缺陷提出变更申请

集成测试工作内容及其工作流程如图 3-19 所示。

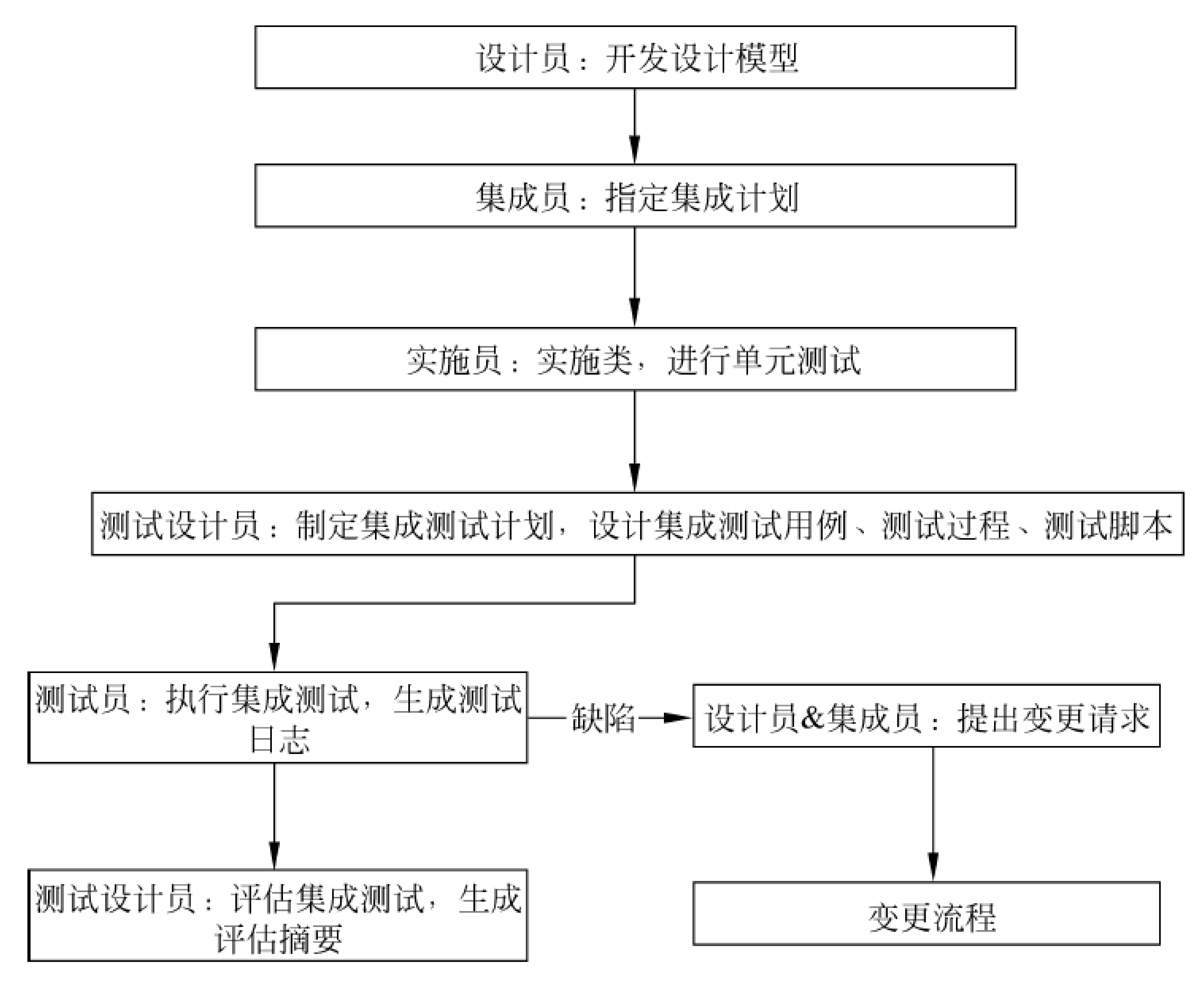


图 3-19　集成测试工作流程

4）集成测试产生的工作清单

（1）软件集成测试计划。

（2）集成测试用例。

（3）测试过程。

（4）测试脚本。

（5）测试日志。

（6）测试评估摘要。

2．集成测试常用方案选型

集成测试的实施方案有很多种，如自底向上集成测试、自顶向下集成测试、Big-Bang 集成测试、三明治集成测试、核心集成测试、分层集成测试、基于使用的集成测试等。下面是实践中证实有效的集成测试方案。

1）自底向上集成测试

自底向上的集成（Bottom-Up Integration）方式是最常使用的方法。其他集成方法都或多或少地继承、吸收了这种集成方式的思想。自底向上集成方式从程序模块结构中最底层的模块开始组装和测试。因为模块是自底向上进行组装的，对于一个给定层次的模块，它的子模块（包括子模块的所有下属模块）事先已经完成组装并经过测试，所以不再需要编制桩模块（一种能模拟真实模块，给待测模块提供调用接口或数据的测试用软件模块）。自底向上集成测试的步骤大致如下。

步骤一：按照概要设计规格说明，明确有哪些被测模块。在熟悉被测模块性质的基础上对被测模块进行分层，在同一层次上的测试可以并行进行，然后排出测试活动的先后关系，制定测试进度计划。图 3-20 给出了自底向上的集成测试过程中各测试活动的拓扑关系。利用图论的相关知识，可以排出各活动之间的时间序列关系，处于同一层次的测试活动可以同时进行，而不会相互影响。

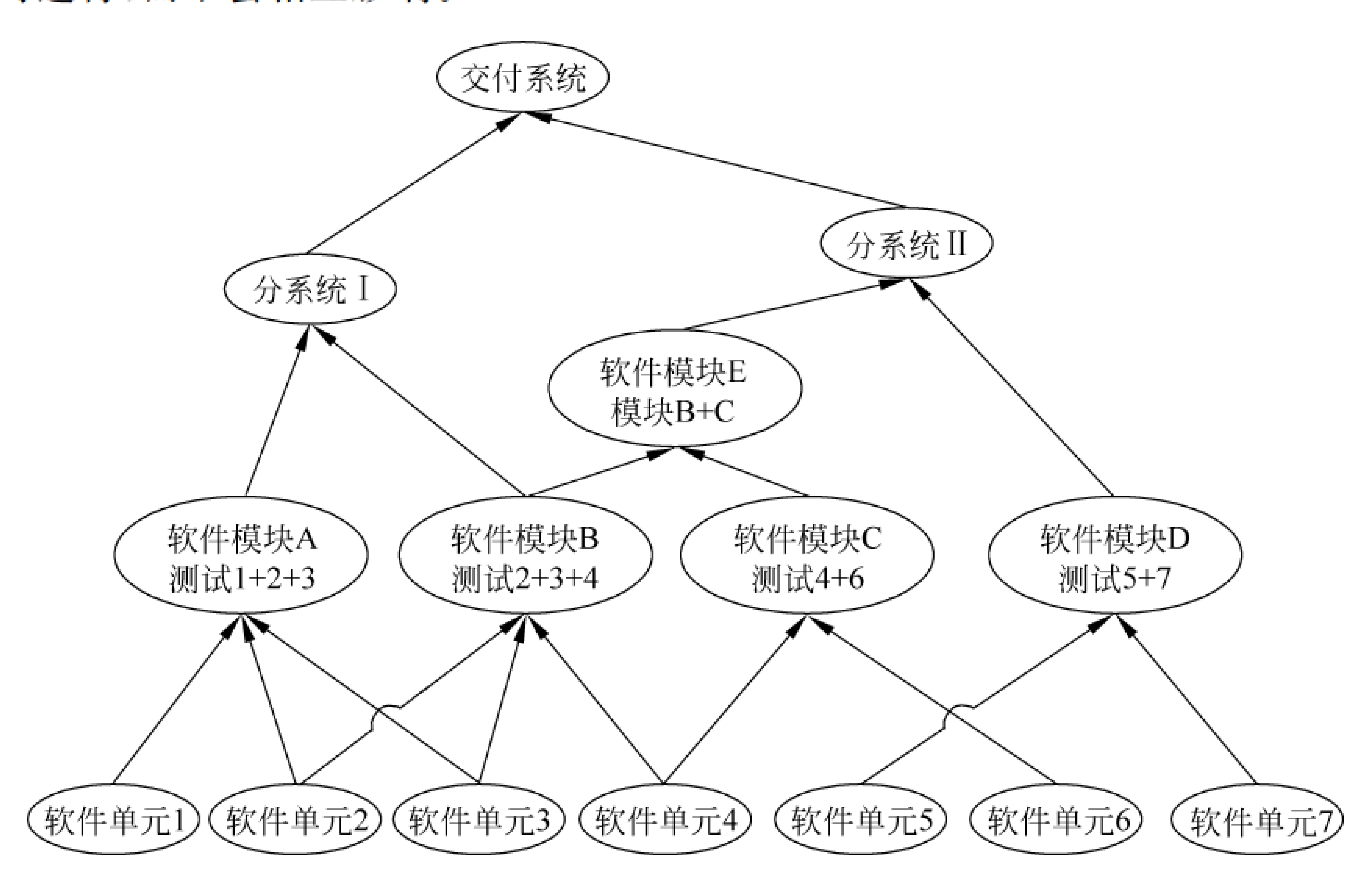


图 3-20　测试方案选型

步骤二：在步骤一的基础上，按时间线序关系，将软件单元集成为模块，并测试在集成过程中出现的问题。这里，可能需要测试人员开发一些驱动模块来驱动集成活动中形成的被测模块。对于比较大的模块，可以先将其中的某几个软件单元集成为子模块，然后再集成为一个较大的模块。

步骤三：将各软件模块集成为子系统（或分系统）。检测各自子系统是否能正常工作。同样，可能需要测试人员开发少量的驱动模块来驱动被测子系统。

步骤四：将各子系统集成为最终用户系统，测试是否存在各分系统能否在最终用户系统中正常工作。

方案点评：自底向上的集成测试方案是工程实践中最常用的测试方法。相关技术也较为成熟。它的优点很明显：管理方便，测试人员能较好地锁定软件故障所在位置。但它对于某些开发模式不适用，如使用 XP 开发方法，它会要求测试人员在全部软件单元实现之前完成核心软件部件的集成测试。尽管如此，自底向上的集成测试方法仍不失为一个可供参考的集成测试方案。

2）核心系统先行集成测试

核心系统先行集成测试法的思想是先对核心软件部件进行集成测试，在测试通过的基础上再按各外围软件部件的重要程度逐个集成到核心系统中。每次加入一个外围软件部件都产生一个产品基线，直至最后形成稳定的软件产品。核心系统先行集成测试法对应的集成过程是一个逐渐趋于闭合的螺旋形曲线，代表产品逐步定型的过程。其步骤如下。

步骤一：对核心系统中的每个模块进行单独的、充分的测试，必要时使用驱动模块和桩模块。

步骤二：对于核心系统中的所有模块一次性集合到被测系统中，解决集成中出现的各类问题。在核心系统规模相对较大的情况下，也可以按照自底向上的步骤，集成核心系统的各组成模块。

步骤三：按照各外围软件部件的重要程度以及模块间的相互制约关系，拟定外围软件部件集成到核心系统中的顺序方案。方案经评审以后，即可进行外围软件部件的集成。

步骤四：在外围软件部件添加到核心系统以前，外围软件部件应先完成内部的模块级集成测试。

步骤五：按顺序不断加入外围软件部件，排除外围软件部件集成中出现的问题，形成最终的用户系统。

方案点评：该集成测试方法对于快速软件开发很有效果，适合较复杂系统的集成测试，能保证一些重要的功能和服务的实现。缺点是采用此法的系统一般应能明确区分核心软件部件和外围软件部件，核心软件部件应具有较高的耦合度，外围软件部件内部也应具有较高的耦合度，但各外围软件部件之间应具有较低的耦合度。

3）高频集成测试

高频集成测试是指同步于软件开发过程，每隔一段时间对开发团队的现有代码进行一次集成测试。如某些自动化集成测试工具能实现每日深夜对开发团队的现有代码进行一次集成测试，然后将测试结果发到各开发人员的电子邮箱中。该集成测试方法频繁地将新代码加入到一个已经稳定的基线中，以免集成故障难以发现，同时控制可能出现的基线偏差。使用高频集成测试需要具备一定的条件：可以持续获得一个稳定的增量，并且该增量内部

已被验证没有问题；大部分有意义的功能增加可以在一个相对稳定的时间间隔(如每个工作日)内获得；测试包和代码的开发工作必须是并行进行的，并且需要版本控制工具来保证始终维护的是测试脚本和代码的最新版本；必须借助于使用自动化工具来完成。高频集成一个显著的特点就是集成次数频繁，显然，人工的方法是不胜任的。

高频集成测试一般采用如下步骤来完成。

步骤一：选择集成测试自动化工具。如很多Java项目采用JUnit＋Ant方案来实现集成测试的自动化，也有一些商业集成测试工具可供选择。

步骤二：设置版本控制工具，以确保集成测试自动化工具所获得的版本是最新版本。如使用CVS进行版本控制。

步骤三：测试人员和开发人员负责编写对应程序代码的测试脚本。

步骤四：设置自动化集成测试工具，每隔一段时间对配置管理库的新添加的代码进行自动化的集成测试，并将测试报告汇报给开发人员和测试人员。

步骤五：测试人员监督代码开发人员及时关闭不合格项。

按照步骤三至步骤五不断循环，直至形成最终软件产品。

方案点评：该测试方案能在开发过程中及时发现代码错误，能直观地看到开发团队的有效工程进度。在此方案中，开发维护源代码与开发维护软件测试包被赋予了同等的重要性，这对有效防止错误、及时纠正错误都很有帮助。该方案的缺点在于测试包有时候可能不能暴露深层次的编码错误和图形界面错误。

以上介绍了几种常见的集成测试方案，一般来讲，在现代复杂软件项目集成测试过程中，通常采用核心系统先行集成测试和高频集成测试相结合的方式进行，自底向上的集成测试方案在采用传统瀑布式开发模式的软件项目集成过程中较为常见。实践中应该结合项目的实际工程环境及各测试方案适用的范围进行合理的选型。

3.5　系统测试

系统测试是将已经确认的软件、计算机硬件、外设、网络等其他元素结合在一起，进行信息系统的各种组装测试和确认测试，系统测试是针对整个产品系统进行的测试，目的是验证系统是否满足了需求规格的定义，找出与需求规格不符或与之矛盾的地方，从而提出更加完善的方案。系统测试发现问题之后要经过调试找出错误原因和位置，然后进行改正。基于系统整体需求说明书的黑盒类测试，应覆盖系统所有联合的部件。对象不仅包括需测试的软件，还要包含软件所依赖的硬件、外设甚至包括某些数据、某些支持软件及其接口等。

1. 非功能性测试

1）安全性测试

软件安全性轻则造成操作的不方便，重则造成数据的破坏或丢失甚至系统的崩溃和人身的安全，因此，软件安全性是一个不容忽视的重要问题。我们可以简单地把软件的安全性作为一个或多个特定的功能来考虑，从而在软件生命周期的早期就加以考虑。

为了帮助设计一个安全的信息系统，在产品设计的最开始就必须注意安全的问题，例如需求中应有安全性的相关项目、设计和代码评审应有专门针对安全性的内容等，然后才是测试。

测试员仅能测试验证软件的安全性。当然,对于没有在软件需求书上标明的可能影响系统运行安全的隐性需求测试人员也要努力地发现,这也是一个有经验的安全性测试人员的可贵之处。

当然,理论上没有任何一个信息系统是安全的,因为只要进行攻击,任何系统都能被攻破,只不过付出的代价的大小不同。而一般说某个信息系统是安全的就是基于如果要攻破该系统所必须付出的代价要高于或远远高于攻破系统后获得的利益。

软件安全性测试策略如表 3-14 所示。

表 3-14　软件安全性测试策略

测试目标	测试应用或系统的安全机制,保证系统运行和使用的安全性
测试策略	采取静态分析技术和功能测试两种方式拦截系统开发时存在的漏洞 软件生命周期早期的代码、设计评审(由开发人员完成)对软件安全性的提高非常有效和重要,可以人工评审和自动化工具结合的方法进行
测试重点考虑	• 相关信息安全法律法规的要求 • 测试环境 • 网络安全 • 日志评审 • 文件完整性检查 • 系统软件安全 • 客户端应用安全 • 客户端到服务端应用通信安全 • 服务端应用安全
测试通过准则	应用满足和符合采用的安全机制,在应用规定的程度上能保证系统正常运行和安全使用

软件安全性测试包括应用软件、网络系统、数据库和系统软件安全性测试。根据系统安全指标不同测试策略也不同。

(1) 用户认证安全的测试要考虑问题:

① 系统中是否有不同的用户使用权限;

② 系统会不会因用户的权限的改变造成混乱;

③ 系统中会不会出现用户冲突;

④ 用户登录密码是否可见、可复制;

⑤ 是否可以通过绝对路径登录系统(复制用户登录后的链接直接进入系统);

⑥ 用户退出系统后是否删除了所有鉴权标记,是否可以使用后退键而不通过输入口令进入系统。

(2) 应用方面安全的测试要考虑问题:

① 缓冲区溢出;

② 无效的数据类型;

③ 关键信息是否采用加密技术;

④ 是否可以通过绝对路径进入系统;

⑤ 使用的端口号;

⑥ 远程进入服务(如有);

⑦ 文件完整性检查;

⑧ 日志评审；

⑨ 模拟黑客攻击。

(3) 系统网络安全的测试要考虑问题：

① 物理连接上的安全，包括无线部分(如果有)；

② 防火墙、防病毒软件的安全；

③ 测试采取的防护措施是否正确装配好，有关系统的补丁是否打上；

④ 模拟非授权攻击，看防护系统是否坚固；

⑤ 采用成熟的网络漏洞检查工具检查系统相关漏洞；

⑥ 入侵网络监察系统和防护系统；

⑦ 采用各种木马检查工具检查系统木马情况；

⑧ 采用各种防外挂工具检查系统各组程序的外挂漏洞。

(4) 数据库安全考虑问题：

① 系统数据是否机密；

② 系统数据的完整性；

③ 系统数据可管理性；

④ 系统数据的独立性；

⑤ 系统数据可备份和恢复能力(数据备份是否完整，可否恢复，恢复是否完整)。

(5) 系统级别软件安全考虑问题：

① 系统级别软件(如操作系统、数据库系统和中间件系统等)是否是最新的，尤其如果使用开源软件或免费软件；

② 系统软件是否有相应的补丁，尤其是安全性方面的补丁包；

③ 从安全性考虑，系统级别软件是否是最可靠的(如使用 Tomcat 还是 WebLogic)；

④ 从安全性考虑，系统级别软件是否匹配应用设计技术。

2) 安装测试

安装测试有两个目的。第一个目的是确保软件能够在不同的条件下进行安装，例如，首次安装、升级安装、完整或自定义安装；在正常和异常条件下的安装。异常条件例如磁盘空间不足、安装用户缺少目录创建权限等。第二个目的是验证软件在安装后能正确操作。这通常意味着要运行大量为功能测试开发的测试用例。

建议对于安装测试，其重点应该放在第一个目的上，对于第二个目的可以根据需要和实际情况穿插在后续的测试，如冒烟测试、功能测试中体现。

软件安装测试策略如表 3-15 所示。

表 3-15　软件安装测试策略

测试目标	验证被测软件在各种需要的软硬件配置下正确安装，包括下列情况： • 新安装。新的机器，之前从未安装过该软件 • 更新。该机器之前安装过该软件同样的版本 • 更新。该机器之前安装过该软件的老版本
测试策略	手工或自动化 不管是手工安装还是自动化脚本安装，都要严格按照用户安装手册规定的操作步骤进行

续表

测试重点考虑	有没有安装手册 对于第二个目的的安装测试，怎样选取预先规定的功能测试的子集
测试通过准则	对于第一个目的：确保软件能够在不同的条件下进行安装。成功的安装应该是在整个安装过程中和安装完成后没有任何异常错误出现，能成功登录该应用 对于第二个目的：验证软件在安装后能正确操作。这个需要运行预先规定的一功能测试的子集

3）配置和兼容性测试

配置和兼容性测试验证被测软件在不同的软硬件配置下的操作和表现。

在大多数生产环境中，特定的硬件规格，如客户端工作站、网络情况、服务器配置甚至型号都会不同。另外，不同客户端工作站上也许有不同的软件负载，如不同的应用软件、不同的浏览器、浏览器的不同版本的插件、驱动程序等；甚至不同的操作系统，如 Windows XP，Windows Vista 等；服务器软件也会不一致，如数据库软件，如 Oracle，Microsoft SQL Server、中间件软件等。

该类测试对产品显得尤其重要，对于项目来说也有巨大的指导作用，如项目经理为了最优化配置（从成本和实用的角度等），就需要进行不同软硬件情况下的配置和兼容性测试。该类测试也往往和性能测试相结合以决定在不同软硬件情况下的最优化配置（如从成本和实用的角度等）。

软件配置和兼容性测试策略如表 3-16 所示。

表 3-16 软件配置和兼容性测试策略

测试目标	验证被测软件在各种需要的软硬件配置下能正确运行
测试策略	使用功能测试脚本。首先，针对特定硬件配置下： • 打开和关闭许多和被测软件无关的、但和被测软件共同在同一机器上的其他许多软件，如同机上的 Microsoft 应用软件，Excel 或 Word 等（如果有），可以在测试过程中和测试前就打开 • 在此基础上执行相应的功能测试
测试重点考虑	硬件配置情况复杂多样，使用哪些硬件配置呢？建议最好做些调研，如考虑目标用户或最终用户的使用习惯和经济承受能力 打开哪些和被测软件无关的、但和被测软件共同在同一机器上的其他许多软件？建议最好做些调研，如考虑目标用户或最终用户会使用哪些其他的常用软件？尤其在使用目标软件时往往同时会打开或使用的软件
测试通过准则	在某一特定的硬件配置下，在许多和被测软件无关的、但和被测软件共同在同一机器上的其他许多软件运行或使用情况下，相应的功能测试没有失败

4）易用性测试

易用性是人类工程学的目标。软件的易用性是一个比较有特点的问题，会随着具体产品或项目的特征和要求而有巨大差异，例如，手机软件和一般 Windows 平台下的软件易用性相差很大，又如一核反应堆的关闭序列和一语音信箱的菜单系统的易用性有着天壤之别。即使对于同一个软件，不同的用户也会有不同的感受。当然，也不是说对软件的易用性就毫无测试办法和标准，对于同一类软件还是有它的通用性可言的。因此下面尝试从一些比较通用的方面讨论它。

易用性主要考虑软件使用时人的因素和感受，因此先介绍两个概念。用户与程序相互交互的介质称为用户接口（User Interface，UI），用户接口对不同软件种类也会有巨大的差异，对于 PC 来说，现在有了精致的完善的图形用户接口（Graphical User Interface，GUI）。很快人们就可以对着 PC 听和说，那时又会有新的 UI。

另外，Accessibility Testing 也属于易用性测试范畴。这种测试一般针对对软件有特殊要求的群体，例如部分残疾人。

软件易用性测试策略如表 3-17 所示。

表 3-17　软件易用性测试策略

测试目标	考虑软件使用时人的因素和感受
测试策略	使用功能测试脚本 对于一些比较通用的规则可以做成检查表的形式
测试重点考虑	• 遵守通用的标准和指导方针。如 Windows 平台下的软件应和 Windows 平台的风格一致，而 Mac 平台下的软件显然应该和 Mac 平台的风格一致 • 直觉的。如用户界面是干净的、不多余的，用户界面是经过良好组织和布局的 • 一致性。如快捷键和菜单选项间；术语和命名；“确认”和“取消”按钮的位置布局等 • 灵活性。如用户希望能手工输入、粘贴、插入文件内容等方式作为输入方法 • 舒服的。如较长时间的等待应有进度条提示，一般情况下程序应有较快的反应。但也不是绝对的，举个例子，高性能就一定好吗？未必如此，例如当快速到连提示信息或出错信息都不能看清时用户反而会觉得很不舒服 • 正确的。例如拼写；图标是否有同样的大小和背景；还有是否所见即所得（What You See Is What You Get，WYSIWYG） • 有用的。没有大量的过量功能，这个特征尤其在产品上表现突出，用户往往要面对大量的无用的或者不需要的功能
测试通过准则	该类测试对于不同产品和项目有较大的差异性，因此尤其要加强对该类测试用例的评审，在业务人员、设计人员等一致同意的基础上进行测试，测试人员也要关注软件需求中应有该方面的相关描述

5）数据和数据库完整性测试

在整个系统测试过程中，数据库和数据库过程应作为系统的子系统进行测试。这些测试不使用用户界面作为数据进入界面进行测试。对不同的数据库管理系统，可能存在不同的工具和技术，如 Oracle 和 Microsoft SQL Server。

软件数据和数据库完整性测试策略如表 3-18 所示。

表 3-18　软件数据和数据库完整性测试策略

测试目标	保证数据存取方法和过程正确运行，在整个数据操作过程中和结束后没有数据破坏
测试策略	使用有效的和无效的数据或数据的请求调用各个数据库存取方法和过程 检查数据库保证数据正如预期地进入，所有的数据库事件正确地发生，或者检查数据保证对于正确的请求取得正确的数据
测试重点考虑	测试也许需要一个 DBMS 开发环境或驱动程序以便在数据库中直接进入或更改数据 确定哪些数据存取方法和数据过程重点测试 测试的先后顺序
测试通过准则	所有的数据库存取方法和过程正如设计要求一样运行，没有任何的数据破坏

6）接口测试

在集成测试过程中，接口测试显得特别重要，首先要确认被测软件是否集成了规定的单元或子系统、外部系统（如果有）；其次，对相应的接口要进行详细的测试；最后也应该对集成后的所有功能进行相应的测试以确保集成后整个系统功能的正确性。

软件接口测试策略如表 3-19 所示。

表 3-19 软件接口测试策略

测试目标	确保“测试需求”中对应的所有工作版本的内部单元组合到一起后能够按照设计的意图协作运行，接口的调用正确
测试策略	使用功能测试脚本 按照设计规范，对所有接口设计相应的测试用例进行手工或自动化测试，包括系统内部接口和外部接口，尤其注意和第三方的软硬件接口（如果有）
测试重点考虑	测试的顺序性 数据是否能正确传递 接口之间的调用 状态的变化是否正确（如果有） 接口之间的异常处理功能
测试通过准则	所有与接口有关的测试用例功能没有失败

7）文档测试

文档也是发布的软件产品的重要组成部分，因此也应该对文档进行相应的测试，尤其对安装手册（前面已有所述）、用户使用手册和在线帮助手册要进行重点测试。正确的文档能减少维护费用和提高软件的可维护性、改善软件易用性、减少责任等。

软件文档测试策略如表 3-20 所示。

表 3-20 软件文档测试策略

测试目标	保证发布的软件产品中的文档的正确性
测试策略	按照手册描述，逐章节阅读、检查和验证相应的内容，例如，严格地按照用户使用手册上的描述去操作和使用程序
测试重点考虑	与用户相关的文档的测试（如安装手册、用户使用手册和在线帮助手册） 术语的正确性和一致性 是否覆盖了产品的所有方面 功能描述的准确性 图形和屏幕截屏是否正确 举例是否正确 拼写和语法 超文本链接
测试通过准则	文档所有内容描述准确和正确

8）失效恢复测试

失效恢复测试保证被测软件能成功地从硬件、软件或网络故障中失效恢复而不会有数据或事务的丢失。

对于必须时刻保持运行的系统，失效备援测试保证当一个失效备援条件发生时，替代或

备份系统正确地接管失败的系统而不会有数据或事务的丢失。

恢复测试是一种对立的测试过程，让应用或系统遭受极端或仿真条件来引起一个故障，例如设备的输入、输出错误或无效的数据库指针和键。调用恢复过程、监控和检查应用或系统来验证已完成正确的应用、系统和数据恢复。

软件失效恢复测试策略如表3-21所示。

表3-21 软件失效恢复测试策略

测试目标	验证恢复过程（手工或自动化）正确恢复数据库、应用和系统到期望的、已知的状态
测试策略	手工或自动化测试 如电源中断、通信中断、网络中断、操作中断等可以手工直接执行；操作中断，尤其是有关数据库的操作、网络操作等也可以使用一些自动化工具执行
测试重点考虑	测试设计或执行中应考虑： • 客户端电源中断 • 服务端电源中断 • 客户端和服务器之间通信中断 • 网络异常问题，如瞬间的断开 • 关键功能执行过程中操作异常中断 • 备份系统有效性（如果有） • 数据过滤过程中断、数据同步过程中断等 • 无效的数据库指针或键 • 数据库中无效的或被破坏的数据元素
测试通过准则	在上面所有的测试中，当恢复程序完成时，应用、数据库和系统应该成功返回到期望的、已知的状态

2. 性能测试

提到软件性能测试的时候，有一点是很明确的：测试关注的重点是“性能”。那么，本书要解决的第一个问题就是：究竟什么是“软件性能”？

一般来说，性能是一种指标，表明软件系统或构件对于其及时性要求的符合程度；其次，性能是软件产品的一种特性，可以用时间来进行度量。

性能的及时性用响应时间或者吞吐量来衡量。响应时间是对请求做出响应所需要的时间。

对于单个事务，响应时间就是完成事务所需的时间；对于用户任务，响应时间体现为端到端的时间。例如，“用户单击OK按钮后两秒内收到结果”就是一个对用户任务响应时间的描述，具体到这个用户任务中，可能有多个具体的事务需要完成，每个事务都有其单独的响应时间。

对交互式的应用（例如典型的Web应用）来说，一般以用户感受到的响应时间来描述系统的性能，而对非交互式应用（嵌入式系统或是银行等的业务处理系统）而言，响应时间是指系统对事件产生响应所需要的时间。

通常，对软件性能的关注是多个层面的：用户关注软件性能，管理员关注软件性能，产品的开发人员也关注软件性能，那么这些不同的关注者所关注的“性能”的具体内容是不是都完全相同呢？如果不同，这些不同又在哪里？最后，作为软件性能测试工程师，不同层面

的软件性能都需要关注，在关注全部这些层面的性能体现的时候，又应该注意哪些内容呢？下面从三个不同层面来对软件性能进行阐述。

1）用户视角的软件性能

从用户的角度来说，软件性能就是软件对用户操作的响应时间。说得更明确一点，对用户来说，当用户单击一个按钮、发出一条指令或是在 Web 页面上单击一个链接，从用户单击开始到应用系统把本次操作的结果以用户能察觉的方式展示出来，这个过程所消耗的时间就是用户对软件性能的直观印象。图 3-21 以一个 Web 系统为例，说明了用户的这种印象。

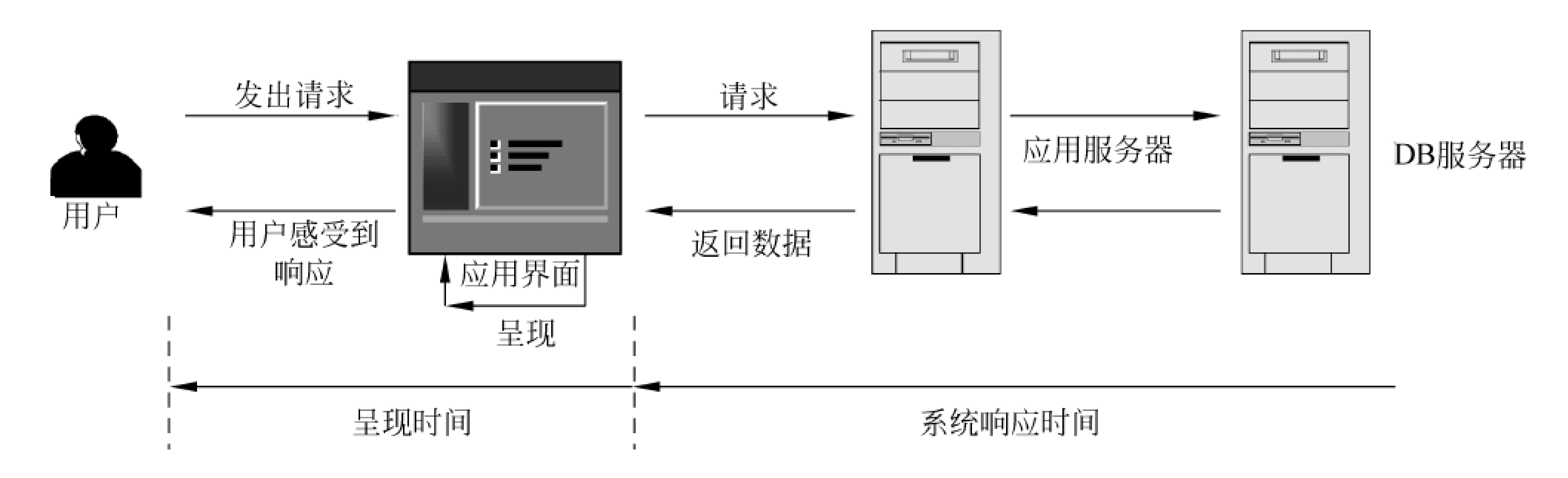


图 3-21 Web 系统的响应

必须要说明的是，用户所体会到的“响应时间”既有客观的成分，也有主观的成分。例如，用户执行了某个操作，该操作返回大量数据，从客观的角度来说，事务的结束应该是系统返回所有的数据，响应时间应该是从用户操作开始到所有数据返回完成的整个耗时；但从用户的主观感知来说，如果采用一种优化的数据呈现策略，当少部分数据返回之后就立刻将数据呈现在用户面前，则用户感受到的响应时间就会远远小于实际的事务响应时间（顺便说一下，这种技巧是在 C/S 结构的管理系统中开发人员常用的一种技巧）。

2）管理员视角的软件性能

从管理员的角度来看，软件系统的性能首先表现在系统的响应时间上，这一点和用户视角是一样的。但管理员是一种特殊的用户，和一般用户相比，除了会关注一般用户的体验之外，他还会关心和系统状态相关的信息。例如，管理员已经知道，在并发用户数为 100 时，A 业务的响应时间为 8s，那么此时的系统状态如何呢？服务器的 CPU 使用是不是已经达到了最大值？是否还有可用的内存？应用服务器的状态如何？设置的 JVM 可用内存是否足够？数据库的状况如何？是否还需要进行一些调整？这些问题普通的用户并不关心，因为这不在他们的体验范围之内；但对管理员来说，要保证系统的稳定运行和持续的良好性能，就必须关心这些问题。

另一方面，管理员还会想要知道系统具有多大的可扩展性，处理并发的能力如何；而且，管理员还会希望知道系统可能的最大容量是什么，系统可能的性能瓶颈在哪里，通过更换哪些设备或是进行哪些扩展能够提高系统性能，了解这些情况，管理员才能根据系统的用户状况制定管理措施，在系统出现计划之外的用户增长等紧急情况的时候能够立即制定相应措施，进行迅速的处理；此外，管理员可能还会关心系统在长时间的运行中是否足够稳定，是否能够不间断地提供业务服务等。

因此，从管理员的视角来看，软件性能绝对不仅仅是应用的响应时间这么一个简单的

问题。

表 3-22 给出了管理员关注的部分性能相关问题的列表。

表 3-22　管理员关注的部分性能相关问题

管理员关心的问题	软件性能描述
服务器的资源使用状况合理吗	资源利用率
应用服务器和数据库的资源使用状况合理吗	资源利用率
系统是否能够实现扩展	系统可扩展性
系统最多能支持多少用户的访问？系统最大的业务处理量是多少	系统容量
系统性能可能的瓶颈在哪里	系统可扩展性
更换哪些设备能够提高系统性能	系统可扩展性
系统能否支持 7×24h 的业务访问	系统稳定性

3）开发视角的软件性能

从开发人员的角度来说，对软件性能的关注就更加深入了。开发人员会关心主要的用户感受——响应时间，因为这毕竟是用户的直接体验；另外，开发人员也会关心系统的扩展性等管理员关心的内容，因为这些也是产品需要面向的用户（特殊的用户）。但对开发人员来说，其最想知道的是“如何通过调整设计和代码实现，或是如何通过调整系统设置等方法提高软件的性能表现”，以及“如何发现并解决软件设计和开发过程中产生的由于多用户访问引起的缺陷”，因此，其最关注的是使性能表现不佳的因素和由于大量用户访问引发的软件故障，也就是人们通常所说的“性能瓶颈”和系统中存在的在大量用户访问时表现出来的缺陷。

举例来说，对于一个没有达到预期性能规划的应用，开发人员最想知道的是，这个糟糕的性能表现究竟是由于系统架构选择的不合理还是由于代码实现的问题引起？由于数据库设计的问题引起？抑或是由于系统的运行环境引发？

或者，对于一个即将发布到现场给用户使用的应用，开发人员可能会想要知道当大量用户访问这个系统时，系统会不会出现某些故障，例如，是否存在由于资源竞争引起的挂起？是否存在由于内存处理等问题引起的系统故障？

因此，对开发人员来说，单纯获知系统性能“好”或者“不好”的评价并没有太大的意义，他们更想知道的是“哪些地方是引起不好的性能表现的根源”或是“哪里可能存在故障发生的可能”。

表 3-23 给出了开发视角的软件性能关注内容。

表 3-23　开发人员关注的性能问题

开发人员关心的问题	问题所属层次
架构设计是否合理	系统架构
数据库设计是否存在问题	数据库设计
代码是否存在性能方面的问题	代码
系统中是否有不合理的内存使用方式	代码
系统中是否存在不合理的线程同步方式	设计与代码
系统中是否存在不合理的资源竞争	设计与代码

4）SEI 负载测试计划过程

SEI 负载测试计划过程（SEI Load Testing Planning Process）是一个关注于负载测试计划的方法，其目标是产生“清晰、易理解、可验证的负载测试计划”。SEI 负载测试计划过程包括 6 个关注的区域（Area）：目标、用户、用例、生产环境、测试环境和测试场景。

SEI 负载测试计划过程将以上述 6 个区域作为负载测试计划需要重点关注和考虑的内容，其重点关注以下几个方面的内容。

（1）生产环境与测试环境的不同。由于负载测试环境与实际的生产环境存在一定的差异，因此，在测试环境上对应用系统进行的负载测试结果很可能不能准确反映该应用系统在生产环境上的实际性能表现，为了规避这个风险，必须仔细设计测试环境。

（2）用户分析。用户是对被测应用系统性能表现最关注和受影响最大的对象，因此，必须通过对用户行为进行分析，依据用户行为模型建立用例和场景。

（3）用例。用例是用户使用某种顺序和操作方式对业务过程进行实现的过程，对负载测试来说，用例的作用主要在于分析和分解出关键的业务，判断每个业务发生的频度、业务出现性能问题的风险等。

从 SEI 负载测试计划过程的描述中可以看到，SEI 负载测试计划过程给出了负载测试需要关注的重点区域，但严格来说，其并不能被称为具体的方法论，因为其仅给出了对测试计划过程的一些关注内容，而没有能够形成实际的可操作的过程。

同功能测试一样，性能测试也必须经历测试需求、测试设计、测试执行、测试分析等阶段，但由于性能测试自身的特殊性（例如，需要引入工具，分析阶段相对重要），性能测试过程又不能完全套用功能测试过程。

SEI 负载测试计划过程在负载测试需要关注的具体内容上提供了参考，但其并不是一个完整的测试过程。

5）RBI 方法

RBI（Rapid Bottleneck Identify）方法是 Empirix 公司提出的一种用于快速识别系统性能瓶颈的方法。该方法基于以下一些事实。

（1）发现的 80％系统的性能瓶颈都由吞吐量制约；

（2）并发用户数和吞吐量瓶颈之间存在一定的关联；

（3）采用吞吐量测试可以更快速定位问题。

RBI 方法首先访问服务器上的“小页面”和“简单应用”，从应用服务器、网络等基础的层次上了解系统吞吐量表现；其次选择不同的场景，设定不同的并发用户数，使其吞吐量保持基本一致的增长趋势，通过不断增加并发用户数和吞吐量，观察系统的性能表现。

在确定具体的性能瓶颈时，RBI 将性能瓶颈的定位按照一种“自上而下”的分析方式进行分析，首先确定是由并发还是由吞吐量引发的性能表现限制，然后从网络、数据库、应用服务器和代码本身 4 个环节确定系统性能具体的瓶颈。

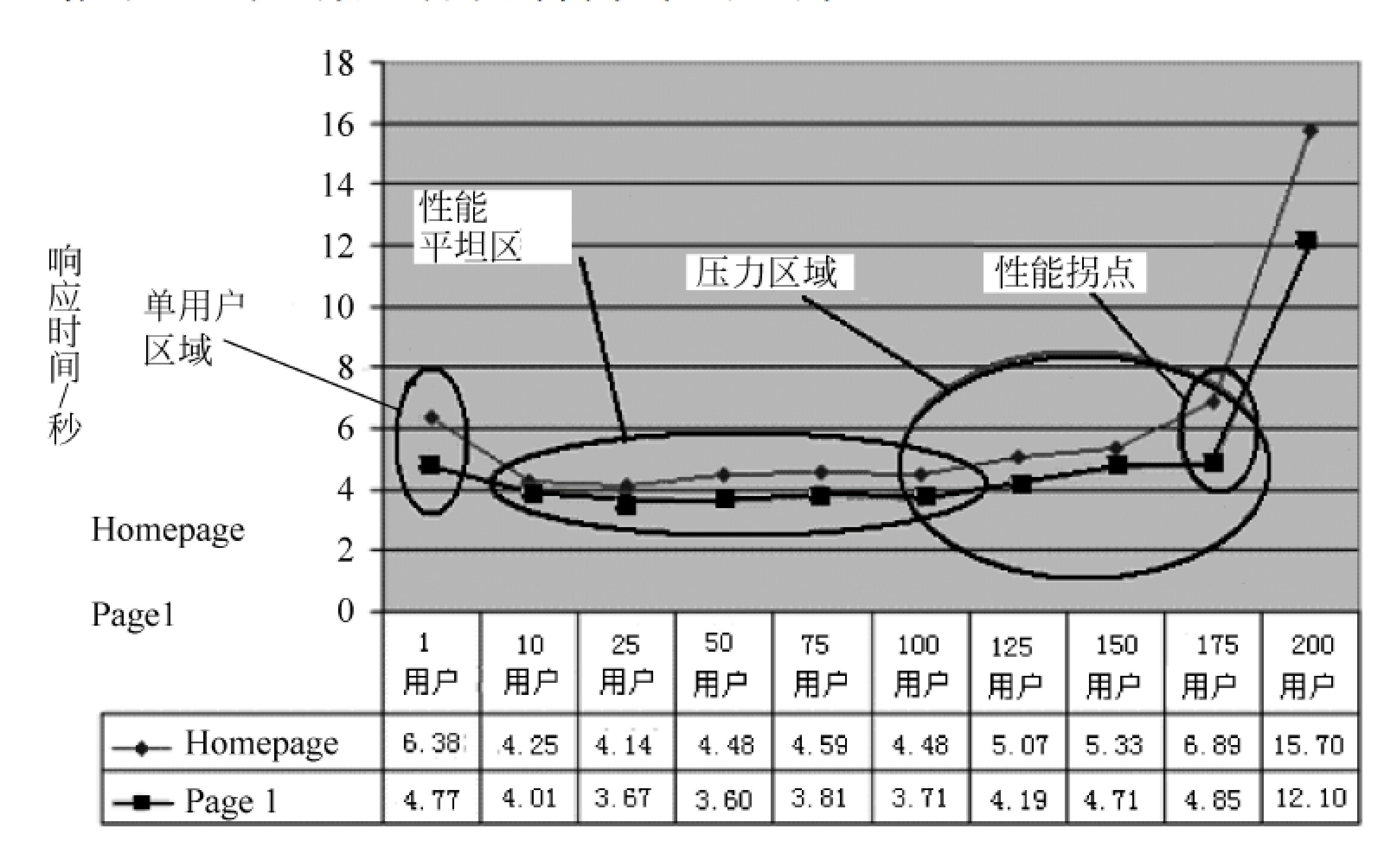
RBI 方法在性能瓶颈的定位过程中能发挥良好的作用，其对性能分析和瓶颈定位的方法值得借鉴，但其也不是完整的性能测试过程。

6）性能下降曲线分析法

性能下降曲线实际上描述的是性能随用户数增长而出现下降趋势的曲线。而这里所说的“性能”可以是响应时间，也可以是吞吐量或是单击数/秒的数据。当然，一般来说，“性能”

主要是指响应时间。

图 3-22 给出了一个"响应时间下降曲线"的示例。



	1用户	10用户	25用户	50用户	75用户	100用户	125用户	150用户	175用户	200用户
Homepage	6.38	4.25	4.14	4.48	4.59	4.48	5.07	5.33	6.89	15.70
Page 1	4.77	4.01	3.67	3.60	3.81	3.71	4.19	4.71	4.85	12.10

图 3-22　一条典型的响应时间性能下降曲线示例

从图 3-22 可以看到，一条曲线可以分为以下几个部分。

（1）单用户区域——对系统的一个单用户的响应时间。这对建立性能的参考值很有作用。

（2）性能平坦区——在不进行更多性能调优情况下所能期望达到的最佳性能。这个区域可被用作基线或是 benchmark。

（3）压力区域——应用"轻微下降"的地方。典型的、最大的建议用户负载是压力区域的开始。

（4）性能拐点——性能开始"急剧下降"的点。

这几个区域实际上明确标识了系统性能最优秀的区间，系统性能开始变坏的区间，以及系统性能出现急剧下降的点。对性能测试来说，找到这些区间和拐点，也就可以找到性能瓶颈产生的地方。

因此，对性能下降曲线分析法来说，主要关注的是性能下降曲线上的各个区间和相应的拐点，通过识别不同的区间和拐点，从而为性能瓶颈识别和性能调优提供依据。

7）Segue 提供的性能测试过程

图 3-23 给出了 Segue 公司 Silk Performer 提供的性能测试过程。该性能测试过程是一个不断 try-check 的过程。

Silk Performer 提供的性能测试过程从确定性能基线开始，通过单用户对应用的访问获取性能取值的基线，然后设定可接受的性能目标（响应时间），用不同的并发用户数等重复进行测试。

Segue 提供的这种性能测试方法非常适合性能调优和性能优化，通过不断重复的 try-check 过程，可以逐一找到可能导致性能瓶颈的地方并对其进行优化。

但 Segue 提供的这个性能测试过程模型存在与 LoadRunner 的性能测试过程同样的问题，就是过于依赖工具自身，另外，该过程模型缺乏对计划、设计的阶段的明确划分，也没有给出具体的活动和目标。

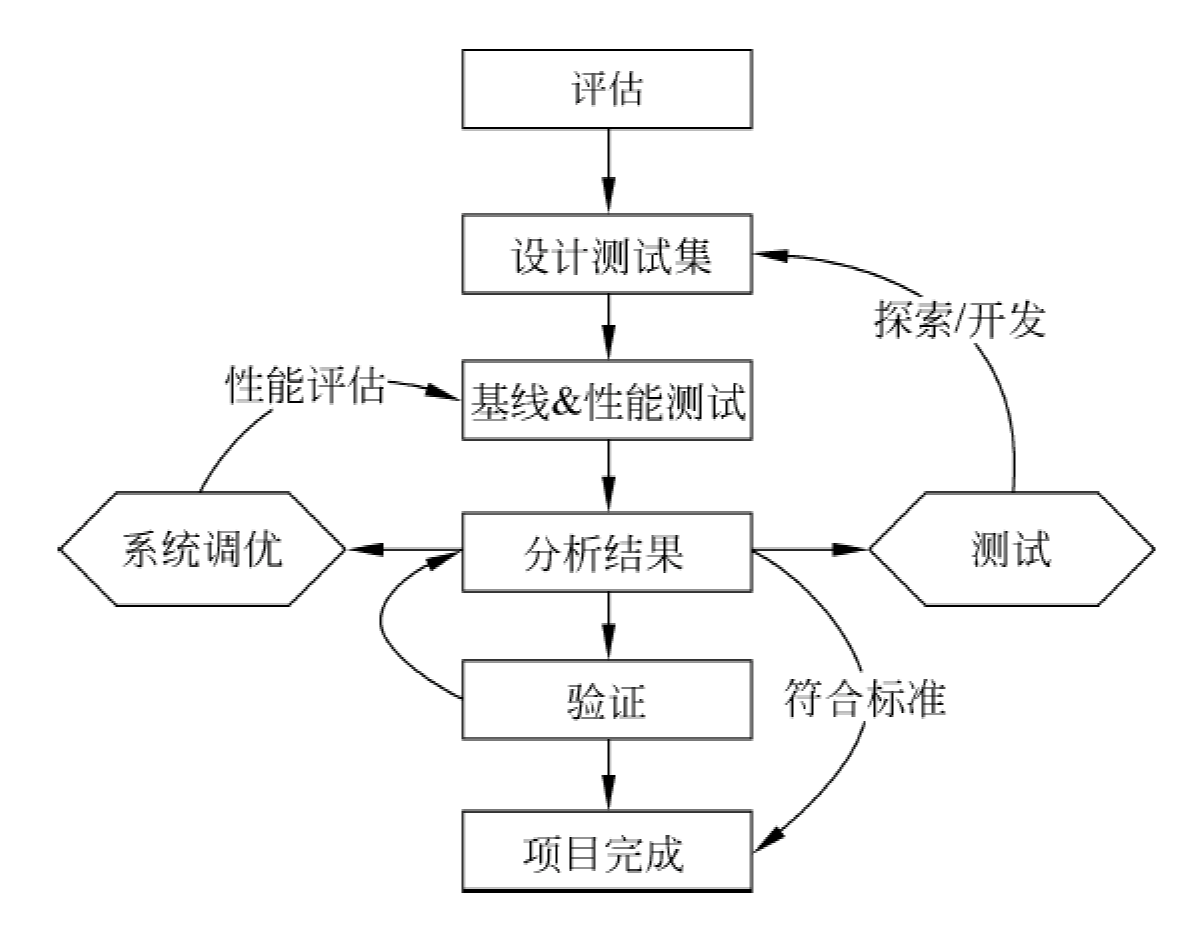


图 3-23　Segue Silk Performer 提供的性能测试过程

思考题

1. 软件测试过程模型有哪几种？它们之间有什么区别？
2. 软件生命周期一般分为哪几个阶段？
3. 常见的软件开发模型有哪几种？各有什么特点？
4. 单元测试、集成测试和系统测试的主要工作内容是什么？
5. 请设计一个单元测试用例。
6. 进行软件安全性测试的策略是什么？

第4章 面向软件工程层面的软件测试

本章学习重点

- 熟悉面向对象测试模型。
- 了解面向对象的各种测试所涉及的内容。
- 了解 AOP 的概念。
- 熟悉 AOP 测试方法。
- 了解 SOA 的概念。
- 熟悉 SOA 测试技术。

本章学习难点

- 熟悉面向对象测试模型。
- 熟悉 AOP 测试方法。
- 熟悉 SOA 测试技术。

4.1 面向对象的测试

面向对象技术是一种全新的软件开发技术,正逐渐代替被广泛使用的面向过程开发方法,被看成是解决软件危机的新兴技术。面向对象技术产生更好的系统结构,更规范的编程风格,极大地优化了数据使用的安全性,提高了程序代码的重用,一些人就此认为面向对象技术开发出的程序无须进行测试。应该看到,尽管面向对象技术的基本思想保证了软件应该有更高的质量,但实际情况却并非如此,因为无论采用什么样的编程技术,编程人员的错误都是不可避免的,而且由于面向对象技术开发的软件代码重用率高,更需要严格测试,避免错误的繁衍。因此,软件测试并没有因面向对象编程的兴起而丧失掉它的重要性。

从 1982 年在美国北卡罗来纳大学召开首次软件测试的正式技术会议至今,软件测试理论迅速发展,并相应出现了各种软件测试方法,使软件测试技术得到极大的提高。然而,一度实践证明行之有效的软件测试对面向对象技术开发的软件多少显得有些力不从心。尤其是面向对象技术所独有的多态、继承、封装等新特点,产生了传统语言设计所不存在的错误

可能性，或者使得传统软件测试中的重点不再显得突出，或者使原来测试经验认为和实践证明的次要方面成为主要问题。例如：

在传统的面向过程程序中，对于函数

```
y = Function(x);
```

只需要考虑一个函数（Function()）的行为特点，而在面向对象程序中，不得不同时考虑基类函数（Base::Function()）的行为和继承类函数（Derived::Function()）的行为。

面向对象程序的结构不再是传统的功能模块结构，作为一个整体，原有集成测试所要求的逐步将开发的模块搭建在一起进行测试的方法已成为不可能。而且，面向对象软件抛弃了传统的开发模式，对每个开发阶段都有不同于以往的要求和结果，已经不可能用功能细化的观点来检测面向对象分析和设计的结果。因此，传统的测试模型对面向对象软件已经不再适用。针对面向对象软件的开发特点，应该有一种新的测试模型。

1. 面向对象测试模型

面向对象的开发模型突破了传统的瀑布模型，将开发分为面向对象分析（Object-Orient Analysis，OOA），面向对象设计（Object-Orient Design，OOD）和面向对象编程（Object-Orient Programming，OOP）三个阶段。分析阶段产生整个问题空间的抽象描述，在此基础上，进一步归纳出适用于面向对象编程语言的类和类结构，最后形成代码。由于面向对象的特点，采用这种开发模型能有效地将分析设计的文本或图表代码化，不断适应用户需求的变动。针对这种开发模型，结合传统的测试步骤的划分，作者建议一种整个软件开发过程中不断测试的测试模型，使开发阶段的测试与编码完成后的单元测试、集成测试、系统测试成为一个整体。测试模型如图 4-1 所示。

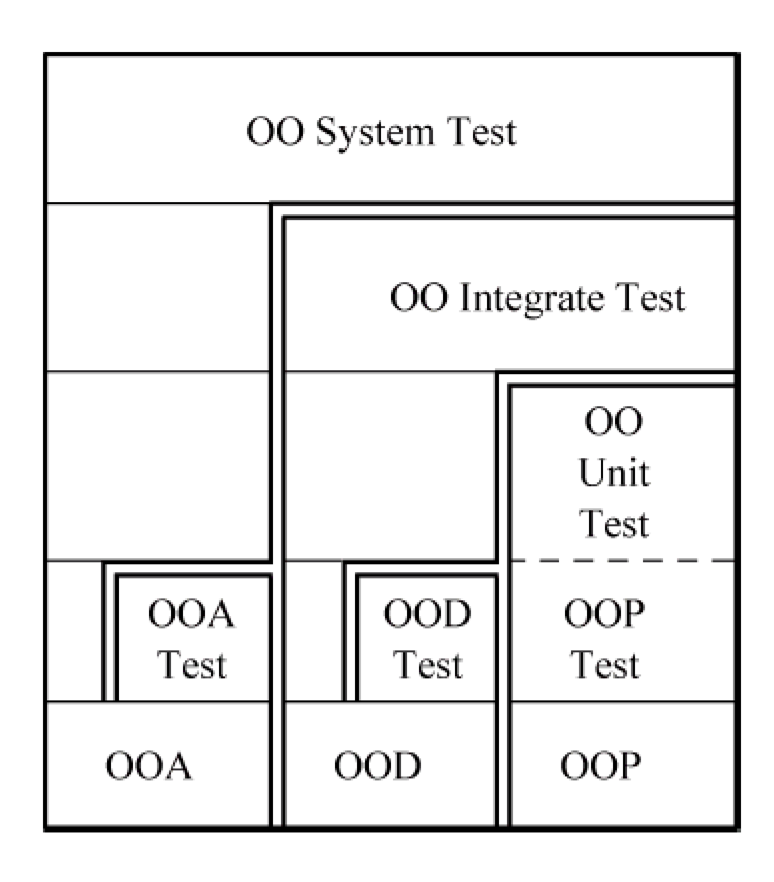


图 4-1　面对对象测试模型

OOA Test 和 OOD Test 是对分析结果和设计结果的测试，主要是对分析设计产生的文本进行，是软件开发前期的关键性测试。OOP Test 主要针对编程风格和程序代码实现进行测试，其主要的测试内容在面向对象单元测试和面向对象集成测试中体现。面向对象单元测试是对程序内部具体单一的功能模块的测试，如果程序是用 C++语言实现，主要就是对类成员函数的测试。面向对象单元测试是进行面向对象集成测试的基础。面向对象集成测试主要对系统内部的相互服务进行测试，如成员函数间的相互作用、类间的消息传递等。面向对象集成测试不但要基于面向对象单元测试，更要参见 OOD 或 OOD Test 结果。面向对象系统测试是基于面向对象集成测试的最后阶段的测试，主要以用户需求为测试标准，需要借鉴 OOA 或 OOA Test 结果。

2. 面向对象分析的测试

传统的面向过程分析是一个功能分解的过程，是把一个系统看成可以分解的功能的集

合。这种传统的功能分解分析法的着眼点在于一个系统需要什么样的信息处理方法和过程，以过程的抽象来对待系统的需要。而面向对象分析是“把E-R图和语义网络模型，即信息造型中的概念，与面向对象程序设计语言中的重要概念结合在一起而形成的分析方法”，最后通常是得到问题空间的图表的形式描述。

OOA直接映射问题空间，全面地将问题空间中实现功能的现实抽象化。将问题空间中的实例抽象为对象（不同于C++中的对象概念），用对象的结构反映问题空间的复杂实例和复杂关系，用属性和服务表示实例的特性和行为。对一个系统而言，与传统分析方法产生的结果相反，行为是相对稳定的，结构是相对不稳定的，这更充分反映了现实的特性。OOA的结果是为后面阶段类的选定和实现，类层次结构的组织和实现提供平台。因此，OOA对问题空间分析抽象的不完整，最终会影响软件的功能实现，导致软件开发后期大量可避免的修补工作；而一些冗余的对象或结构会影响类的选定、程序的整体结构或增加程序员不必要的工作量。因此，本文对OOA的测试重点在其完整性和冗余性。

尽管OOA的测试是一个不可分割的系统过程，为叙述的方便，Coad方法所提出的OOA实现步骤，对OOA阶段的测试划分为以下5个方面。

（1）对认定的对象的测试；

（2）对认定的结构的测试；

（3）对认定的主题的测试；

（4）对定义的属性和实例关联的测试；

（5）对定义的服务和消息关联的测试。

对象、结构、主题等在OOA结果中的位置，如图4-2所示。

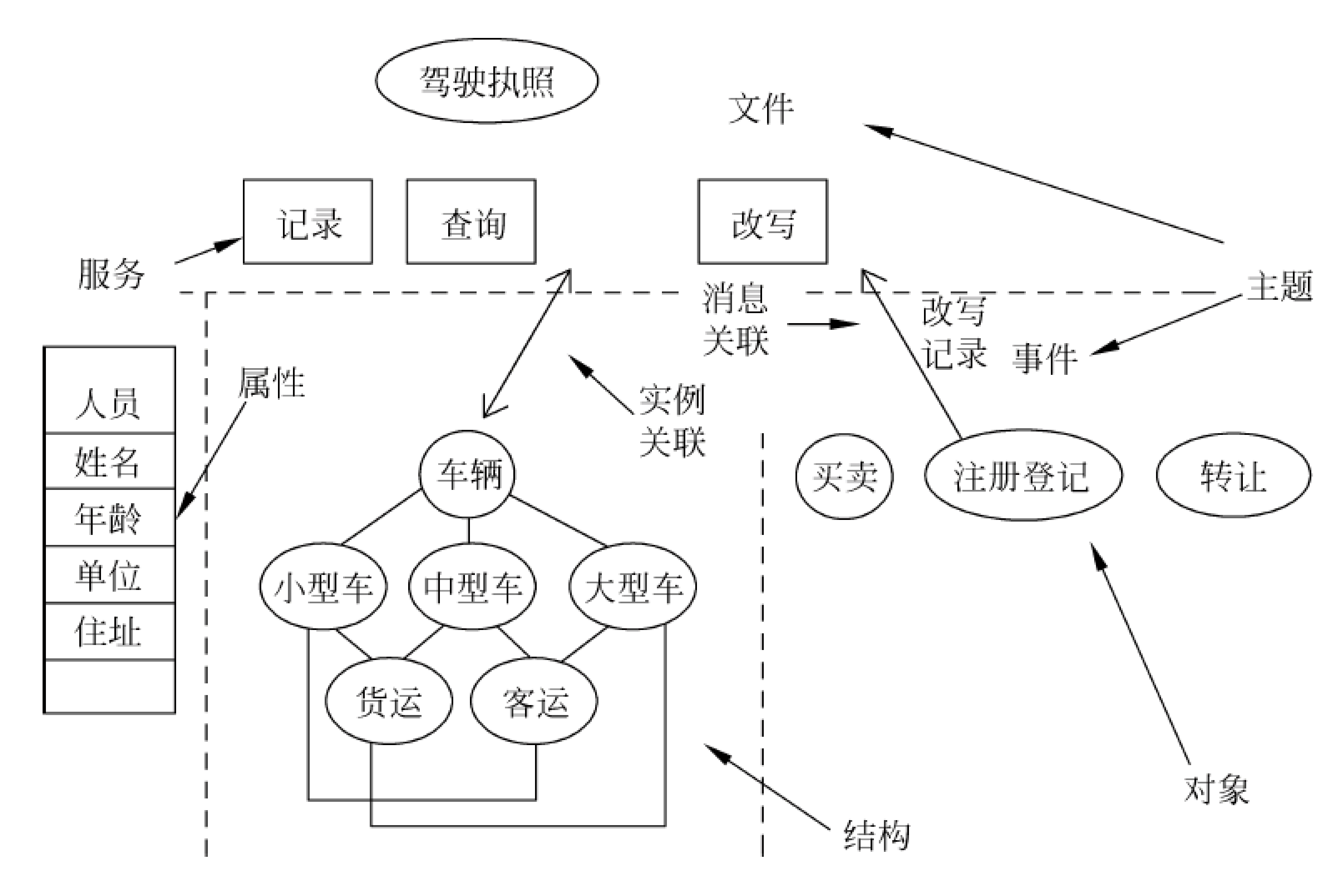


图4-2　车辆管理系统部分OOA分析结果示意图

1）对认定的对象的测试

OOA中认定的对象是对问题空间中的结构、其他系统、设备、被记忆的事件、系统涉及的人员等实际实例的抽象。对它的测试可以从如下方面考虑。

（1）认定的对象是否全面，是否问题空间中所有涉及的实例都反映在认定的抽象对

象中。

(2) 认定的对象是否具有多个属性。只有一个属性的对象通常应看成其他对象的属性,而不是抽象为独立的对象。

(3) 对认定为同一对象的实例是否有共同的,区别于其他实例的共同属性。

(4) 对认定为同一对象的实例是否提供或需要相同的服务,如果服务随着不同的实例而变化,认定的对象就需要分解或利用继承性来分类表示。

(5) 如果系统没有必要始终保持对象代表的实例的信息,提供或者得到关于它的服务,认定的对象也无必要。

(6) 认定的对象的名称应该尽量准确,适用。

2) 对认定的结构的测试

在 Coad 方法中,认定的结构指的是多种对象的组织方式,用来反映问题空间中的复杂实例和复杂关系。认定的结构分为两种:分类结构和组装结构。分类结构体现了问题空间中实例的一般与特殊的关系,组装结构体现了问题空间中实例整体与局部的关系。

对认定的分类结构的测试可从如下方面着手。

(1) 对于结构中的一种对象,尤其是处于高层的对象,是否在问题空间中含有不同于下一层对象的特殊可能性,即是否能派生出下一层对象。

(2) 对于结构中的一种对象,尤其是处于同一低层的对象,是否能抽象出在现实中有意义的更一般的上层对象。

(3) 对所有认定的对象,是否能在问题空间内向上层抽象出在现实中有意义的对象。

(4) 高层的对象的特性是否完全体现下层的共性。

(5) 低层的对象是否有高层特性基础上的特殊性。

对认定的组装结构的测试从如下方面入手。

(1) 整体(对象)和部件(对象)的组装关系是否符合现实的关系。

(2) 整体(对象)的部件(对象)是否在考虑的问题空间中有实际应用。

(3) 整体(对象)中是否遗漏了反映在问题空间中有用的部件(对象)。

(4) 部件(对象)是否能够在问题空间中组装新的有现实意义的整体(对象)。

3) 对认定的主题的测试

主题是在对象和结构的基础上更高一层的抽象,是为了提供 OOA 分析结果的可见性,如同文章对各部分内容的概要。对主题层的测试应该考虑以下方面。

(1) 贯彻 George Miller 的“7+2”原则,如果主题个数超过 7 个,就要求对有较密切属性和服务的主题进行归并。

(2) 主题所反映的一组对象和结构是否具有相同和相近的属性和服务。

(3) 认定的主题是否是对象和结构更高层的抽象,是否便于理解 OOA 结果的概貌。

(4) 主题间的消息联系(抽象)是否代表了主题所反映的对象和结构之间的所有关联。

4) 对定义的属性和实例关联的测试

属性是用来描述对象或结构所反映的实例的特性,而实例关联是反映实例集合间的映射关系。对属性和实例关联的测试从如下方面考虑。

(1) 定义的属性是否对相应的对象和分类结构的每个现实实例都适用。

(2) 定义的属性在现实世界是否与这种实例关系密切。

(3) 定义的属性在问题空间是否与这种实例关系密切。

(4) 定义的属性是否能够不依赖于其他属性被独立理解。

(5) 定义的属性在分类结构中的位置是否恰当,低层对象的共有属性是否在上层对象属性体现。

(6) 在问题空间中每个对象的属性是否定义完整。

(7) 定义的实例关联是否符合现实。

(8) 在问题空间中实例关联是否定义完整,特别需要注意1-多和多-多的实例关联。

5) 对定义的服务和消息关联的测试

定义的服务,就是定义的每一种对象和结构在问题空间所要求的行为。由于问题空间中实例间必要的通信,在OOA中相应需要定义消息关联。对定义的服务和消息关联的测试从如下方面进行。

(1) 对象和结构在问题空间的不同状态是否定义了相应的服务。

(2) 对象或结构所需要的服务是否都定义了相应的消息关联。

(3) 定义的消息关联所指引的服务提供是否正确。

(4) 沿着消息关联执行的线程是否合理,是否符合现实过程。

(5) 定义的服务是否重复,是否定义了能够得到的服务。

3. 面向对象设计的测试

通常的结构化的设计方法,用的是面向作业的设计方法,它把系统分解以后,提出一组作业,这些作业是以过程实现系统的基础构造,把问题域的分析转化为求解域的设计,分析的结果是设计阶段的输入。

而面向对象设计(OOD)采用“造型的观点”,以OOA为基础归纳出类,并建立类结构或进一步构造成类库,实现分析结果对问题空间的抽象。OOD归纳的类,可以是对象简单的延续,可以是不同对象的相同或相似的服务。由此可见,OOD不是在OOA上的另一思维方式的大动干戈,而是OOA的进一步细化和更高层的抽象。所以,OOD与OOA的界限通常是难以严格区分的。OOD确定类和类结构不仅是满足当前需求分析的要求,更重要的是通过重新组合或加以适当的补充,能方便实现功能的重用和扩增,以不断适应用户的要求。因此,对OOD的测试,作者建议针对功能的实现和重用以及对OOA结果的拓展,从如下三方面考虑。

1) 对认定的类的测试

OOD认定的类可以是OOA中认定的对象,也可以是对象所需要的服务的抽象,对象所具有的属性的抽象。认定的类原则上应该尽量基础性,这样才便于维护和重用。可参考以下一些准则来测试认定的类。

(1) 是否涵盖OOA中所有认定的对象。

(2) 是否能体现OOA中定义的属性。

(3) 是否能实现OOA中定义的服务。

(4) 是否对应着一个含义明确的数据抽象。

(5) 是否尽可能少地依赖其他类。

(6) 类中的方法(C++:类的成员函数)是否单用途。

2）对构造的类层次结构的测试

为能充分发挥面向对象的继承共享特性，OOD的类层次结构，通常基于OOA中产生的分类结构的原则来组织，着重体现父类和子类间一般性和特殊性。在当前的问题空间，对类层次结构的主要要求是能在解空间构造实现全部功能的结构框架。为此，测试如下方面。

（1）类层次结构是否涵盖所有定义的类。

（2）是否能体现OOA中所定义的实例关联。

（3）是否能实现OOA中所定义的消息关联。

（4）子类是否具有父类没有的新特性。

（5）子类间的共同特性是否完全在父类中得以体现。

3）对类库支持的测试

对类库的支持虽然也属于类层次结构的组织问题，但其强调的重点是再次软件开发的重用。由于它并不直接影响当前软件的开发和功能实现，因此，将其单独提出来测试，也可作为对高质量类层次结构的评估。拟订测试点如下。

（1）一组子类中关于某种含义相同或基本相同的操作，是否有相同的接口（包括名字和参数表）。

（2）类中方法（C++：类的成员函数）功能是否较单纯，相应的代码行是否较少（建议不超过30行）。

（3）类的层次结构是否是深度大，宽度小。

4．面向对象编程的测试

典型的面向对象程序具有继承、封装和多态的新特性，这使得传统的测试策略必须有所改变。封装是对数据的隐藏，外界只能通过被提供的操作来访问或修改数据，这样降低了数据被任意修改和读写的可能性，降低了传统程序中对数据非法操作的测试。继承是面向对象程序的重要特点，继承使得代码的重用率提高，同时也使错误传播的概率提高。继承使得传统测试遇见了这样一个难题：对继承的代码究竟应该怎样测试？（参见面向对象单元测试。）多态使得面向对象程序对外呈现出强大的处理能力，但同时却使得程序内“同一”函数的行为复杂化，测试时不得不考虑不同类型具体执行的代码和产生的行为。

面向对象程序是把功能的实现分布在类中。能正确实现功能的类，通过消息传递来协同实现设计要求的功能。正是这种面向对象程序风格，将出现的错误能精确地确定在某一具体的类。因此，在面向对象编程阶段，忽略类功能实现的细则，将测试的目光集中在类功能的实现和相应的面向对象程序风格，主要体现为以下两个方面（假设编程使用C++语言）。

1）数据成员是否满足数据封装的要求

数据封装是数据和数据有关的操作的集合。检查数据成员是否满足数据封装的要求，基本原则是数据成员是否被外界（数据成员所属的类或子类以外的调用）直接调用。更直观地说，当改变数据成员的结构时，是否影响了类的对外接口，是否会导致相应外界必须改动。值得注意，有时强制的类型转换会破坏数据的封装特性。例如：

```
class Hiden
{private:
int a = 1;
```

```
char * p = "hiden"; }
class Visible
{public:
int b = 2;
char * s = "visible"; }
…
…
Hiden pp;
Visible * qq = (Visible * )&pp;
```

在上面的程序段中，pp 的数据成员可以通过 qq 被随意访问。

2）类是否实现了要求的功能

类所实现的功能，都是通过类的成员函数执行。在测试类的功能实现时，应该首先保证类成员函数的正确性。单独地看待类的成员函数，与面向过程程序中的函数或过程没有本质的区别，几乎所有传统的单元测试中所使用的方法，都可在面向对象的单元测试中使用。具体的测试方法在面向对象的单元测试中介绍。类函数成员的正确行为只是类能够实现要求的功能的基础，类成员函数间的作用和类之间的服务调用是单元测试无法确定的。因此，需要进行面向对象的集成测试。具体的测试方法在面向对象的集成测试中介绍。需要着重声明，测试类的功能，不能仅满足于代码能无错运行或被测试类能提供的功能无错，应该以所做的 OOD 结果为依据，检测类提供的功能是否满足设计的要求，是否有缺陷。必要时（如通过 OOD 结果仍不清楚明确的地方）还应该参照 OOA 的结果，以它为最终标准。

5. 面向对象的单元测试

传统的单元测试是针对程序的函数、过程或完成某一定功能的程序块。沿用单元测试的概念，实际测试类成员函数。一些传统的测试方法在面向对象的单元测试中都可以使用，如等价类划分法，因果图法，边值分析法，逻辑覆盖法，路径分析法，程序插装法等。单元测试一般建议由程序员完成。

用于单元级测试进行的测试分析（提出相应的测试要求）和测试用例（选择适当的输入，达到测试要求），规模和难度等均远小于后面将介绍的对整个系统的测试分析和测试用例，而且强调对语句应该有 100%的执行代码覆盖率。在设计测试用例选择输入数据时，可以基于以下两个假设。

（1）如果函数（程序）对某一类输入中的一个数据正确执行，对同类中的其他输入也能正确执行。

（2）如果函数（程序）对某一复杂度的输入正确执行，对更高复杂度的输入也能正确执行。例如，需要选择字符串作为输入时，基于本假设，就无须计较于字符串的长度。除非字符串的长度是要求固定的，如 IP 地址字符串。

在面向对象程序中，类成员函数通常都很小，功能单一，函数间的调用频繁，容易出现一些不宜发现的错误。例如：

（1）if (−1==write (fid, buffer, amount)) error_out();

该语句没有全面检查 write()的返回值，无意中断然假设了只有数据被完全写入和没有写入两种情况。当测试也忽略了数据部分写入的情况，就给程序遗留了隐患。

(2) 按程序的设计,使用函数 strrchr()查找最后的匹配字符,但误程序中写成了函数 strchr(),使程序功能实现时查找的是第一个匹配字符。

(3) 程序中将 if (strncmp(str1,str2,strlen(str1)))误写成了

if (strncmp(str1,str2,strlen(str2)))。如果测试用例中使用的数据 str1 和 str2 长度一样,就无法检测出。

因此,在做测试分析和设计测试用例时,应该注意面向对象程序的这个特点,仔细地进行测试分析和设计测试用例,尤其是针对以函数返回值作为条件判断选择、字符串操作等情况。

面向对象编程的特性使得对成员函数的测试,又不完全等同于传统的函数或过程测试。尤其是继承特性和多态特性,使子类继承或过载的父类成员函数出现了传统测试中未遇见的问题。Brian Marick 给出了以下两方面的考虑。

(1) 继承的成员函数是否都不需要测试?

对父类中已经测试过的成员函数,两种情况需要在子类中重新测试:①继承的成员函数在子类中做了改动;②成员函数调用了改动过的成员函数的部分。

例如,假设父类 Bass 有两个成员函数 Inherited()和 Redefined(),子类 Derived 只对 Redefined()做了改动。Derived::Redefined()显然需要重新测试。对于 Derived::Inherited(),如果它有调用 Redefined()的语句(如 x=x/Redefined()),就需要重新测试;反之,无此必要。

(2) 对父类的测试是否能照搬到子类?

延用上面的假设,Base::Redefined()和 Derived::Redefined()已经是不同的成员函数,它们有不同的服务说明和执行。对此,照理应该对 Derived::Redefined()重新测试分析,设计测试用例。但由于面向对象的继承使得两个函数有相似,故只需在 Base::Redefined()的测试要求和测试用例上添加对 Derived::Redfined()新的测试要求和增补相应的测试用例。例如,Base::Redefined()含有如下语句:

```
If (value < 0) message ("less");
else if (value == 0) message ("equal");
else message ("more");
Derived::Redfined()中定义为
If (value < 0) message ("less");
else if (value == 0) message ("It is equal");
else
{message ("more");
if (value == 88)message("luck"); }
```

在原有的测试上,对 Derived::Redfined()的测试只需做如下改动:将 value==0 的测试结果期望改动;增加 value==88 的测试。

多态有几种不同的形式,如参数多态、包含多态、过载多态。包含多态和过载多态在面向对象语言中通常体现在子类与父类的继承关系,对这两种多态的测试参见上述对父类成员函数继承和过载的论述。包含多态虽然使成员函数的参数可有多种类型,但通常只是增加了测试的繁杂。对具有包含多态的成员函数测试时,只需要在原有的测试分析和基础上扩大测试用例中输入数据的类型的考虑。

6. 面向对象的集成测试

传统的集成测试，是由底向上通过集成完成的功能模块进行测试，一般可以在部分程序编译完成的情况下进行。而对于面向对象程序，相互调用的功能是散布在程序的不同类中，类通过消息相互作用申请和提供服务。类的行为与它的状态密切相关，状态不仅体现在类数据成员的值，也许还包括其他类中的状态信息。由此可见，类相互依赖极其紧密，根本无法在编译不完全的程序上对类进行测试。所以，面向对象的集成测试通常需要在整个程序编译完成后进行。此外，面向对象程序具有动态特性，程序的控制流往往无法确定，因此也只能对整个编译后的程序做基于黑盒子的集成测试。

面向对象的集成测试能够检测出相对独立的单元测试无法检测出的那些类相互作用时才会产生的错误。基于单元测试对成员函数行为正确性的保证，集成测试只关注于系统的结构和内部的相互作用。面向对象的集成测试可以分成两步进行：先进行静态测试，再进行动态测试。

静态测试主要针对程序的结构进行，检测程序结构是否符合设计要求。现在流行的一些测试软件都能提供一种称为"可逆性工程"的功能，即通过原程序得到类关系图和函数功能调用关系图，例如，International Software Automation 公司的 Panorama-2 for Windows 95、Rational 公司的 Rose C++ Analyzer 等，将"可逆性工程"得到的结果与 OOD 的结果相比较，检测程序结构和实现上是否有缺陷。换句话说，通过这种方法检测 OOP 是否达到了设计要求。

动态测试设计测试用例时，通常需要上述的功能调用结构图、类关系图或者实体关系图为参考，确定不需要被重复测试的部分，从而优化测试用例，减少测试工作量，使得进行的测试能够达到一定覆盖标准。测试所要达到的覆盖标准可以是：达到类所有的服务要求或服务提供的一定覆盖率；依据类间传递的消息，达到对所有执行线程的一定覆盖率；达到类的所有状态的一定覆盖率等。同时也可以考虑使用现有的一些测试工具来得到程序代码执行的覆盖率。

具体设计测试用例，可参考下列步骤。

(1) 选定检测的类，参考 OOD 分析结果，确定类的状态和相应的行为，类或成员函数间传递的消息，输入或输出的界定等。

(2) 确定覆盖标准。

(3) 利用结构关系图确定待测类的所有关联。

(4) 根据程序中类的对象构造测试用例，确认使用什么输入激发类的状态、使用类的服务和期望产生什么行为等。

值得注意，设计测试用例时，不但要设计确认类功能满足的输入，还应该有意识地设计一些被禁止的例子，确认类是否有不合法的行为产生，如发送与类状态不相适应的消息，要求不相适应的服务等。根据具体情况，动态的集成测试，有时也可以通过系统测试完成。

7. 面向对象的系统测试

通过单元测试和集成测试，仅能保证软件开发的功能得以实现，但不能确认在实际运行时，它能满足用户的需要，是否大量存在实际使用条件下会被诱发产生错误的隐患。为此，

对完成开发的软件必须经过规范的系统测试。换个角度说，开发完成的软件仅仅是实际投入使用系统的一个组成部分，需要测试它与系统其他部分配套运行的表现，以保证在系统各部分协调工作的环境下也能正常工作。

系统测试应该尽量搭建与用户实际使用环境相同的测试平台，应该保证被测系统的完整性，对临时没有的系统设备部件，也应有相应的模拟手段。系统测试时，应该参考 OOA 分析的结果，对应描述的对象、属性和各种服务，检测软件是否能够完全“再现”问题空间。系统测试不仅是检测软件的整体行为表现，从另一个侧面看，也是对软件开发设计的再确认。

这里说的系统测试是对测试步骤的抽象描述。它体现的具体测试内容如下。

(1) 功能测试：测试是否满足开发要求，是否能够提供设计所描述的功能，是否用户的需求都得到满足。功能测试是系统测试最常用和必需的测试，通常还会以正式的软件说明书为测试标准。

(2) 强度测试：测试系统的能力最高实际限度，即软件在一些超负荷的情况下功能的实现情况。如要求软件某一行为的大量重复、输入大量的数据或大数值数据、对数据库大量复杂的查询等。

(3) 性能测试：测试软件的运行性能。这种测试常常与强度测试结合进行，需要事先对被测软件提出性能指标，如传输连接的最长时限、传输的错误率、计算的精度、记录的精度、响应的时限和恢复时限等。

(4) 安全测试：验证安装在系统内的保护机构确实能够对系统进行保护，使之不受各种非常的干扰。安全测试时需要设计一些测试用例试图突破系统的安全保密措施，检验系统是否有安全保密的漏洞。

(5) 恢复测试：采用人工的干扰使软件出错，中断使用，检测系统的恢复能力，特别是通信系统。恢复测试时，应该参考性能测试的相关测试指标。

(6) 可用性测试：测试用户是否能够满意使用。具体体现为操作是否方便，用户界面是否友好等。

(7) 安装/卸载测试等。

系统测试需要对被测的软件结合需求分析做仔细的测试分析，建立测试用例。

4.2 AOP 测试

面向方面编程(Aspect Orient Programming，AOP)是一种新兴的编程思想，它能够提高软件的模块性和内聚性，降低软件的复杂度，为软件质量的提高提供了一个新的途径。但是，并不能保证程序员不犯错误，而且它本身也不提供正确性验证，那么开发有效的测试方法来检验 AOP 的正确性至关重要。

AOP 是一种关注点分离技术，它运用方面机制捕获横切关注点，将分散的应用组成单独的模块，有效地解决了横切关注点造成的代码交织和散布问题。所谓横切关注点是指横跨系统各个对象层次的模块，但与它所跨越的对象代码在功能上没有相关性。为了描述和实现 Aspect 机制，AOP 引入了一些新的构造：连接点(Joint Point)是程序执行中明确定义的点，在这些点中可执行 Aspect 代码；切入点(Pointcut)是捕获、识别程序中连接点的结

构,是通知的激发条件,它决定了各种 AOP 特征将如何被运用到类中;通知用于声明在切入点表达式中定义的连接点被调用时应执行的动作,它包括 Before、Around 以及 After 三种类型;导言是一个增加方法到存在类中的途径;方面是 AOP 的核心,类似于 OOP 中类的概念,是把切入点和通知封装在一起体现横切关系的模块单元,它也可以包含方法和属性、从其他类或方面扩展以及实现接口等。

与面向过程和面向对象语言不同,AOP 引入了一些新的语言构造,如连接点、切入点、导言以及通知等,它们不仅改变了软件的开发过程,还严重影响了程序的状态和行为(如导言可以把一些新方法和实例变量引入核心关注点中;通知则能够为类引入一些额外的不可见行为),从而导致方面和类之间的交互更加复杂,那么传统的测试方法不能直接应用到 AOP 中,它需要开发专门的测试技术和方法来支持这些特殊构造,这为 AOP 测试带来了困难。

此外,AOP 本身的特性问题也不容忽视:首先方面没有独立的实体或存在物,它们完全依赖其他一些类的上下文;其次,方面的实现同它们的织入上下文紧密关联;再者,方面或类代码中的控制和数据依赖关系不明显;最后就是特殊的织入过程可能还会产生一些突发行为。这些问题都给 AOP 的测试带来了困难。因此,开发适合面向方面的软件测试技术是 AOP 的一项迫切而艰巨的任务。

1. AOP 测试方法

AOP 作为一种新的编程范例,目前还缺乏成熟的测试方法。下面主要介绍 4 种不同的测试方法,它们在实现上各有其特点。

1) 基于错误模型的系统测试方法

任何系统测试方法都是针对程序中出现过的一些错误类型,如果对于错误类型没有任何概念,那么开发的系统测试方法也是没有意义的。AleKander、Bleman 和 Andrews 提出了一个错误模型,它包括 6 种类型的错误,反映了 AOP 与面向对象以及面向过程截然不同的特征。

(1) 错误的切入点结构约束。切入点包含识别、捕捉特定类型连接点的表达式描述。对于一个切入点 p,关注点 C 中每个匹配的连接点 l 将被织入一个与 p 关联的通知。p 表达式中定义的匹配函数用于确定被选择的连接点。如果定义的匹配函数约束过强,一些必需的连接点会被忽略;如果约束太弱,一些本应该忽视的连接点将被捕获。以上任何一种情况都很可能造成织入程序的错误行为,这也是 AOP 中最明显的一类错误。

(2) 错误的方面优先级。在面向方面程序中,多个方面有可能同时影响一个连接点,那么通知的不同织入次序会影响系统的行为。在这种情况下,控制通知织入次序是很重要的,它可以通过为不同的方面指定不同的优先级来确定,高优先级的通知能够被优先织入。

(3) 未能建立期望的后条件。方面会引起类代码中控制流的改变,从而导致类不能实现其类契约的后条件。而其使用者希望关注点能根据它们的契约运行,不管通知是否被织入,方法的后条件都必须满足,也就是在织入过程之后应该保留关注点的行为契约。因此,对于正确的行为,被织入通知必须允许核心关注点的方法满足其后条件。然而,在所有可能的织入上下文以及方面组合中定义不会造成行为契约违背的通知是一个困难的挑战,本身也是一个极可能的错误源。

(4) 未能保存状态不变。关注点的行为是根据其状态的物理表示以及作用于此状态上的方法来定义的。除了要建立方法的后条件外,方法也必须保证其状态不变量的保留。方面的通知和导言能够为关注点引入新的状态,那么要保证织入不会造成状态不变量的违背也是一个艰难的挑战,同时也是一个错误的来源。

(5) 错误的控制流焦点。一个切入点表达式指定了要选择的连接点。然而,这种选择的确定不仅是在织入阶段,很多情况取决于运行时期。因此,要诊断出由于运行时上下文造成的错误是很困难的。

(6) 错误的控制依赖变更。around 通知能够为方法引入新的分支控制流,从而极大地改变了语句之间的依赖,因此要保证程序的正确执行,语句之间的依赖关系也需要测试。虽然这仅仅是一个理论上的错误模型,还没有形成实际完整的测试方法,但是它有助于开发测试工具以及确定测试粗盖策略和标准,为面向方面程序的系统测试做出了重要的第一步。

2) 基于状态的测试方法

从面向对象程序的 FREE 测试设计模型中受到启发,Diahajang XU,Weifeng Xu 和 Kendall Nygard 提出了一种基于状态的测试方法。它的基本思想是:首先,为了详细说明类和方面的状态和行为,把类的 FREE 模型扩展成方面状态模型,然后将 ASM 转化成一棵转换树,该树包含一个可充分测试对象行为以及类与方面之间交互的测试套件。FREE 模型是面向对象软件开发使用的一种状态图,它描述了对象的状态和动态行为,并提供了实现的指导。ASM 模型是 FREE 状态模型的一个扩展,为了表示 AOP 中基本的语言构造——连接点、切入点和通知,A 引认增加了一些新的标志。图 4-3 显示了一个 ASM 例子。其中状态 A 表示初始状态,它接受一个具有约束条件[cond1](值为真时)的消息 M1 后转入状态 B。消息 M1 同时受三个通知影响:具有约束条件[cond2]的 before 通知将 A 转入状态 C;具有约束条件[cond4]的 after 通知将 A 转入状态 E;具有约束条件[cond3]的 Around 通知将 A 转入状态 D。

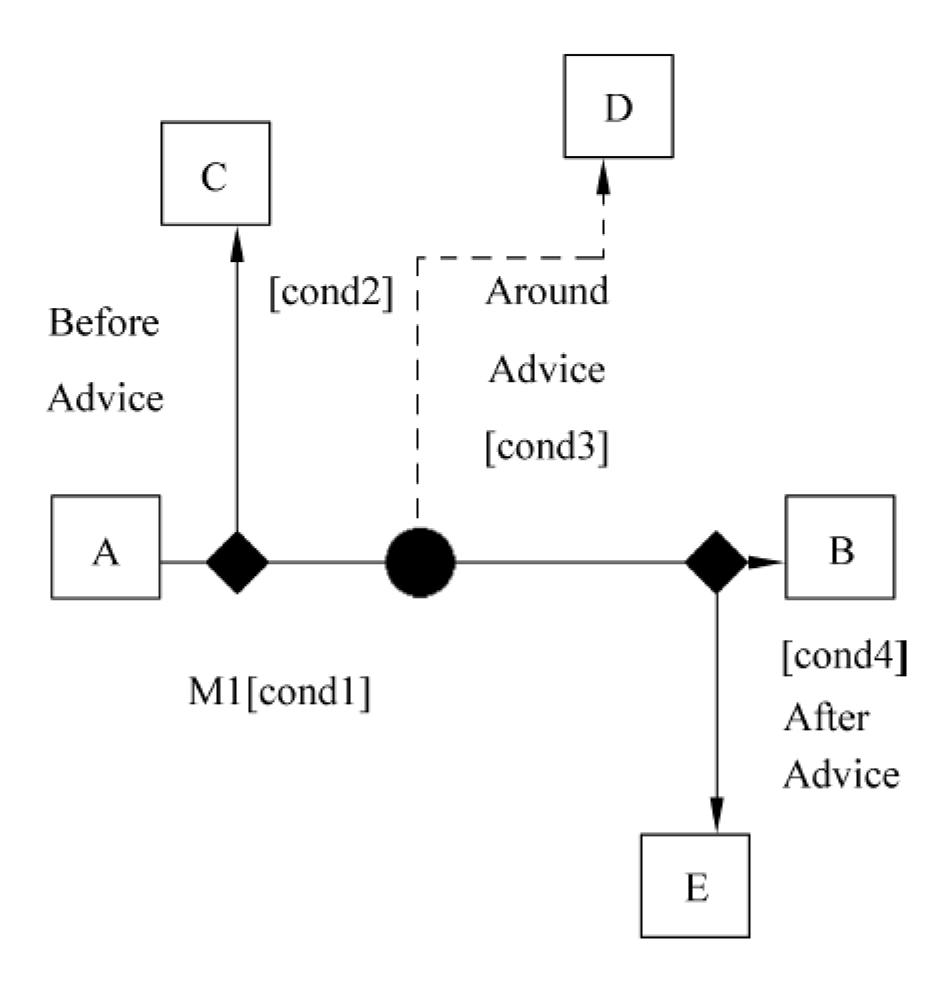


图 4-3 ASM 的例子

转换树测试的核心思想就是通过一定的算法将 ASM 转变成一棵转换树,然后根据从根到叶节点的每条路径所表示的消息序列,生成相应的测试套件。在转换树中,每条从根到叶节点的路径表示检验对象行为的一个测试需求。如果此路径代表的消息序列上的变量被指定为满足相应约束条件的值,那么这个测试需求就变成了一个具体的测试用例。

3) 基于方面流图的测试方法

Weifeng Xu,Dianxiang Xu 和 Vivek Goel 等人提出了一个基于方面流图的测试方法。该方法包括:首先把类状态模型和方面状态模型合并为一个方面域状态模型(Aspect Scope State Model,ASSM);然后,使用通知和方法流图替换 ASSM 中相应的类与方面之间的转化以及对应的动作,从而构造一个方面流图(Aspect Flow Graph,AFG);此外,根据 ASSM 和一组定界参数连同决定行为的参数关系生成一棵转换树;最后,利用 AFG 和转换树,产

生一个基于代码的可运行测试套件。

方面域状态模型 ASSM 的构造是通过合并以下两个状态模型完成的。

(1) 类状态模型：它描述了特定类的状态转换行为。

(2) 方面状态模型：如图 4-4 所示，它包括 4 个状态。a 状态和 ε 状态分别对应方面的入口和出口；其他两个状态依次称为 Action Prior 和 Action Posterior。此外，方面还包括 before 通知、after 通知以及类的 arbltrary 动作，它们是方面中的特定动作。其中，before 通知将方面中的状态从 a 状态转变到 Action Prior 状态；而 after 通知将其从 Action Posterior 转变到 ε 状态；arbltrary 动作则将状态从 Action Prior 变化到 Action Posterior。构造算法的基本思想是检查类状态模型中边表示的每个动作，该动作对应类中一个定义良好的连接点。如果它被方面中的一个切入点捕获，那么使用相应的方面状态模型取代此边。

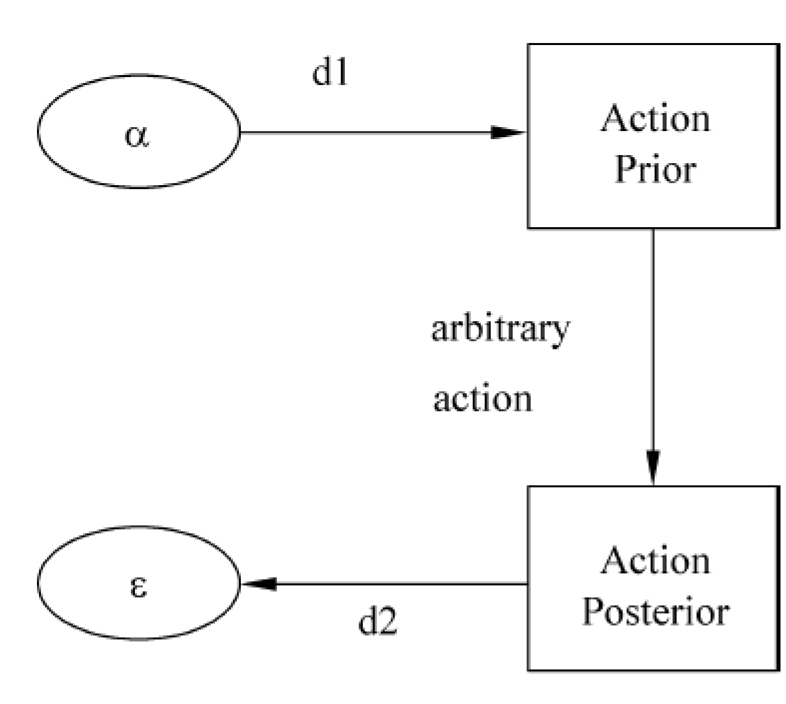


图 4-4 方面状态模型

方面流图类似类流图，仅有的差别在于 AFG 也可以表示由方面引入的控制流。它是通过合并 ASSM 以及方法和通知流图构造的。具体来说，ASSM 中任何导致方面状态改变的动作(通知和方法)都被相应的通知和方法流图替换。AFG 不仅显示了运行时通知(before 和 after)是如何被附加到一个选中的连接点上，也展示了类行为是如何受附加的通知影响。转换树显示了所有可能的状态转换序列，根据这些序列就可以生成满足数据流覆盖标准的测试用例。

4) 其于数据流的单元测试方法

lanjun Zhao 提出一个基于数据流的 AOP 单元测试方法。该方法测试 AOP 中的两类单元：①方面，程序横切实现的模块化单元；②类，行为受一个或多个方面影响。它使用控制流图来寻找被测试单元的 def-use 对，然后利用此信息指导选择测试用例。

"单元测试"的意思就是测试程序的每个单元(基本组件)，从而检验单元的详细设计是否正确实现。然而，它在 AOP 中的含义更难理解。由于方面可以打破类的封装，那么类不再被认为是一个良好封装的内聚单元。此外，一个类可能受多个方面的影响而一个方面也可能影响多个类，方面与类之间的交互也变得很复杂。因此，单独测试 AOP 中的类和方面是不现实的，应该①测试方面时连同那些其行为受通知影响的方法(从方面角度)；②测试类时连同那些能影响该类行为的通知以及能引入新成员到该类中的导言(从类角度)。

为了能正确地测试方面和类，将 AOP 分为以下 5 类。

(1) 聚集方面。它是单个方面连同一个或多个类中的方法，这些方法受方面中通知的影响，表示为 c-aspect。

(2) 聚集类。它是单个类连同一个或多个方面中的通知或导言，通知影响类方法的行为，而导言改变类的类型结构，表示为 c-class。

(3) 聚集方法。它是单个方法连同一个或多个影响其行为的通知，表示为 c-method。

(4) 正常类。它是单个类，行为不受任何方面的影响，表示为 n-class。

(5) 正常方法。它是单个方法，不受任何通知影响，表示为 n-method。

以上每类的流图是单独构造并且其测试方法也是分开执行的。

c-aspect 和 c-class 流图的构造过程分为以下三步。

第一步，构造 c-aspect 和 c-class 的调用图。它是一个有向图，表示 c-aspect 和 c-class 中各模块之间的调用关系。对于 c-aspect 来说，其顶点表示组成 c-aspect 的模块，它们可以是行为受方面影响的 c-method、通知或 n-method。边表示 c-aspect 中模块间的调用次序。

第二步，使用专用顶点（包括框架入口顶点（Frame Entry Vertex）、框架循环顶点（Frame Loop Vertex）、框架出口顶点（Frame Loop Vertex）、框架返回顶点（Frame Return Vertex）和框架调用顶点（Frame Call Vertex））来构造 c-aspect 或 c-class 的框架，并将其插入对应的调用图中。此框架表示 c-aspect 或 c-class 的测试驱动，它可以模拟 c-aspect 或 c-class 中某些模块间的任意调用顺序，支持对 c-aspect 或 c-class 的数据流分析。

第三步，用模块的控制流图替换调用图中的相应顶点。三步产生的结果就是一个称为框架式控制流图（Framed Control Flow Graph）的 c-aspect 图。c-class 的 FCFG 构造方法类似。

对于每个待测试的类或方面，此方法执行一个三层测试策略。

（1）模块内测试。仅测试单个模块。对于 c-aspect，其模块可以为一个 c-method、n-method 或 c-aspect 的导言；对于 c-class，其模块可以是 c-class 的一个 n-method 或 c-method。它只检验模块体内出现的 def-use 对。

（2）模块间测试。测试一个公共模块连同其在一个方面或类中直接或间接调用的其他一些模块。它要测试涉及多个模块的 def-use 对。

（3）方面内/类内测试。检验那些可以在方面或类外被访问并能够被方面或类的使用者以任意次序调用的模块。不同的调用次序将检验不同的 def-use 对。

2. AOP 测试方法的对比分析

下面简单地总结各个测试方法的特点，同时使用错误模型比较各方法的测试性能。

（1）基于状态的测试方法捕获方面对类状态模型的影响。它运用转换树来测试 AOP 中的所有路径，能够达到 $N+$ 的测试覆盖率。它可以显示程序中所有的状态控制错误、潜在路径以及许多误用的状态错误。同时，该方法还可以反映错误模型中的两类错误——错误的切入点结构约束和未能保存状态不变量。然而，此方法经常会遇到状态爆炸的问题，因此，如何确定最感兴趣或最重要的路径是值得研究的下一步工作。

（2）基于方面流图的测试方法构造方面流图来描述方面与类之间的交互。这有助于运用多种白盒测试方法以及测试覆盖标准，而且如果类代码已经被测试，那么只要把注意力集中在由方面引入的新行为和交互上，就可以重用类的测试用例并且能够更多地降低测试成本。此外，该方法还可以检测错误模型中的一类错误——错误的控制依赖变更。

（3）基于数据流的单元测试方法通过组合单元测试和数据流测试技术来检验 AOP 中的类和方面。它没有针对任何特定的错误类型，但能反映错误模型中的第 4 类错误——未能保存状态不变量，因为状态错误通常是与错误的数据流相关的。不过，该方法只针对方面和类本身，没有考虑其集成测试问题，也没有考虑检验扩展类和方面的测试方法。

表 4-1 总结了三个测试方法与错误类型之间的关系。由此可以得出：已提出的测试方法只针对 AOP 中出现的局部错误类型，测试覆盖面太窄，检验效率不高，缺乏全面的考虑。

AOP 测试技术要不断发展，必须要在测试方法的深度和广度上进行研究。

表 4-1　测试方法与错误类型之间的关系

	基于状态的测试	基于方面流图的测试	基于数据流的单元测试
1	否	是	否
2	否	否	否
3	否	否	否
4	是	是	否
5	否	否	否
6	否	否	是

4.3　SOA 测试

面向服务软件架构（Service Oriented Architecture，SOA）是一种新型的软件架构思想，同面向对象思想的出现一样，带来了对软件测试的新的挑战。SOA 在更高的层次上实现了软件复用和封装，通过业务流程的方式将原来独立的软件应用系统结合在一起共同完成一系列的功能，这极大地方便了企业信息整合。但是，这种方便带来的测试量是巨大的，测试的方法和侧重点也与单个软件应用系统有很大的区别。

SOA 是一种架构模型，它可以根据需求通过网络对松散耦合的粗粒度应用组件进行分布式部署、组合和使用。服务层是 SOA 的基础，可以直接被应用调用，从而有效控制系统中与软件代理交互的人为依赖性。SOA 本质上是服务的集合。服务间彼此通信，这种通信可能是简单的数据传送，也可能是两个或更多的服务协调进行某些活动，服务间需要某些方法进行连接。而所谓服务就是精确定义、封装完善、独立于其他服务所处环境和状态的函数。

SOA 同面向对象概念有一个共同目的，就是为了提高软件复用的程度。面向对象的软件采用了信息隐蔽、封装和继承、多态和动态绑定等技术，这些都为软件测试带了新的课题。SOA 通过将单个系统中某种业务处理封装为服务，实现了在更高层次上的软件复用，信息更加隐蔽。同时 SOA 作为一个体系架构而不是一款应用软件，其复杂性不是单一应用软件可以比拟的。

1. 面向服务架构对软件测试的影响

SOA 测试的三个维度包括服务、进程和性能。

1）服务测试

服务级测试是最重要的，因为核心服务是 SOA 的基础。然而，各种服务在编写方面彼此之间差别很大，因为它们的开发者是不同的，有些服务粒度也许比较粗，而有些服务可能会很细，还有一些服务则可能设计粗劣，还有些服务也许会建立在现有界面和 API 上，因此它们就更加复杂，更需要进行质量保证试验，因为需要在一个中间层之外再加一个中间层。这其中并没有什么技巧，主要就是验证各项服务的用途、界面功能是否正确以及验证 WSDL 和规划等内容。另外，还要考虑 design-time 和 run-time 的诊断情况，确保解决那些

重要的概念、功能和回归测试。服务测试相当于单元测试，在这个阶段，需要对服务接口、局部数据结构的完整性、边界条件、覆盖条件和出错处理等进行测试。

2）业务流程测试

除了服务级测试之外，还必须测试服务被加入到业务流程中和混合应用中的方式。因为这些一般都是以服务本身的方式显示出来的，这只是对核心服务以及服务调节单位和系统关系的整个进程的另一个级别的测试。这与测试面向对象式系统非常相似，但是这些系统有异构开发以及 run-time 平台，因此复杂性更强。业务流程测试相当于集成测试/系统测试。在这个阶段，可以根据业务流程的特点，观察系统级的输入和输出是否符合预期。

3）基础设施测试

SOA 的实施除了需要具体的应用软件系统提供的服务外，还需要一系列的基础设施才能正常运行，这些基础设施一般包括服务注册和发现系统（一般为 UDDI）、授权系统等。这些系统也是以独立的接口提供服务，需要进行服务测试。由于这些系统只有在业务流程中才能真正发挥作用，也需要将其放在业务流程中进行测试。

基础设施类似于一般软件应用系统中调用的 API、SDK。但是，由于 SOA 的应用针对性很强，不同的 SOA 架构基础设施不可能完全一样，如授权系统，由于涉及的系统不同，不同的 SOA 之间肯定存在差异，所以在测试时不能假定基础设施的正确性，必须进行测试。

4）性能测试

性能测试也很重要，SOA 是一个松散耦合的系统架构，其中业务流程通过调用不同软件应用系统提供的服务完成，这些应用系统本身的性能可能因为设计或被分配计算、处理任务的多寡而差别很大，一个应用系统的处理速度慢会造成整个 SOA 系统架构的瓶颈。SOA 的大部分质量问题都跟性能有关。SOA 中的性能测试就是对服务、构成、业务流程和系统等不同级别的测试问题。在测试系统的整体性能时，必须沿着体系结构对数据流图中最高层到最底层进行分解，找出系统中存在问题的组件。

SOA 中的服务可能封装了普通应用软件系统中的一系列处理过程，如移动提供的短信发送服务就可能封装了用户认证、欠费查询、短信发送、计费等多个模块，而业务流程由于集成了大量服务，性能方面可能暴露的问题更大。有实验数据表明，SOA 的性能在目前情况下可能只相当于不采用 SOA 时的 10%，所以性能测试对于 SOA 应用的意义超过了以前任何一种软件测试。通过性能测试，可以找到瓶颈服务从而对其进行调优，也可以根据各个组成服务的性能特点在保证处理逻辑的情况下对业务流程进行调优，这些都会带来 SOA 性能的极大提高。

2. SOA 测试技术

1）服务的功能性测试

SOA 中的业务通过组合多个应用软件系统提供的服务而形成，从 SOA 的范畴来看，服务是组成 SOA 业务流程的有机模块，对服务的测试相当于单元测试，但是从服务对应的应用系统来说，一个服务可能通过调用应用的多个模块而完成了一系列的功能，对服务的测试除了对服务本身输入输出的正确性（黑盒）进行测试外，还应该包括更深层次对服务所涉及的应用软件系统模块进行测试，在这个意义上，对 SOA 中具体服务的测试实际上是对提供服务的应用软件系统与该服务相关的部分进行集成测试。

具体说来，对服务的功能性测试主要包括以下几个方面。

（1）输入有效性测试

由于服务调用同函数调用类似，但也有着很大不同，函数调用如果输入错误的参数类型或参数个数，在编译时一般就能被编译器发现。而服务一般采用远程调用方式，其参数通过某种规范的方式发布出来，但是没有很好的外在机制发现调用者的非法调用，这就需要服务本身提供较强的错误发现能力。这种能力也加大了输入有效性测试的任务。一般测试的内容包括参数类型、参数个数测试，同时对于一些特定的服务调用方式，如以 Web Services 方式发布的服务，还存在字符编码测试等。

（2）输入边界值测试

边界值是软件最容易出现错误的地方，对于服务来说也是测试重点。

2）业务流程分析和测试

业务流程是 SOA 实现具体功能的有机单元，对其的测试可以采用类似传统软件测试中的数据流分析和测试的方法。一个业务流程中可能存在多个分支，测试需要保证每个分支都能被充分测试到。下面从几个方面来说明。

（1）测试用例的设计

业务流程是一个抽象的概念，从本质上说，它定义了服务调用的次序、约束条件以及输入值等，从这个概念上说，业务流程与面向对象方法中的类相似。对应地，由于对象是类的实例化，受测试的也是具体的对象。在对业务流程的测试中，也需要定义一个对应的测试实例，即测试用例。

设计测试用例主要内容是设计测试的输入条件，使得最终得到的测试用例集可以覆盖业务流程的每个分支。由于业务流程有着比较明确的定义（如使用 BPEL 定义的 BP），且目前来说其逻辑结构不算很复杂，做到路径的全覆盖不是太难。但是由于业务流程中有的分支被调用的约束条件部分来自于前驱服务的调用结果，这就需要对每个服务的输入输出进行分析，从而设计出有效的覆盖输入。

（2）测试结果的分析

同传统的测试一样，集成测试结果不一定就是数据输出，有可能是 GUI 上的某个标志，也可能只是内部状态的一个改变。在 SOA 的业务流程测试中，这种情况更是发挥了极致，一个业务流程测试流程的执行会造成参与此业务的各个应用软件系统的状态的改变。一个结果的正确与否不能仅判断流程中最后一环的正确执行与否，而是需要去分析参与此业务流程的各个应用软件系统的运行状态。这需要在设计测试用例的时候就要预测到各个系统状态的预期值。

3）基础架构测试

前面已经提到过，基础架构是 SOA 得以运行的必备组件。这一块的测试必须针对每一个具体实施来进行，不存在移植后就不需测试的侥幸。例如授权系统，当每一个新的应用系统加入这个架构，都必须进行重新测试，即使这个新的应用系统采用的是同原有系统中相同的认证方式，因为对其的测试不仅是测试服务和业务流程本身，也是对新系统提供的服务接口的测试。

对基础架构的测试也可分为服务测试和业务流程测试两块。前者测试基础架构作为一个单独系统的功能完整性，后者则是业务流程中必不可少的一个测试条件。

4）性能测试

性能是 SOA 发展道路上的一大障碍，SOA 提供了灵活的业务组合机制以及通用的服务调用方式（如 Web Service），但是带来的牺牲之一便是性能的下降。SOA 性能测试的目的性很强，一是发现性能瓶颈，从而对瓶颈部分进行调优；二是对业务流程的逻辑结构进行调优，从而提高整体性能。从性能测试来说可以包括以下几个方面。

（1）服务性能测试

通过对单个服务的压力测试，可以知道此服务的最大负载、响应时间等参数，从而作为以后判断瓶颈的依据。

（2）网络流量测试

由于 SOA 是针对具体的环境进行实施，要能预测要实施地点的实际网络带宽，通过测试业务相互调用的流量测试，尽量将大流量的调用放在带宽较大的地点，而对于较远程的调用则尽量采用各种减小流量的方法，如数据库采用存储过程、XML 采用压缩等方法。

（3）业务流程性能测试

通过测试业务流程的单位时间执行次数，并根据其中业务逻辑和对服务性能测试的结果判断出瓶颈，如果瓶颈服务可以调优则对其进行优化，如果由于某些原因（业务系统不能动或是开发人员的离开）无法对单个服务进行优化，则需要对业务的逻辑进行优化，如通过任务的调度，将瓶颈服务从关键路径中调离转为并发（如果可以的话）。

（4）服务器、服务器软件等外围条件测试

服务的发布和调用一般由专用的服务器软件完成，如 Web Service 可以通过 IBM 公司的 Web Sphere、BEA 公司的 WebLogic 或是开源的 Tomcat 来发布，这些服务器软件的执行性能各不相同，在选用时需要对其进行一些测试，了解其性能中的并发数、响应时间等具体参数，结合需要的实施的 SOA 的具体情况进行选择。同时服务器的性能也是一个重要指标，往往瓶颈的出现不是因为软件的问题而是应用服务器出了问题。

思考题

1. 面向对象的软件测试包括哪几种？
2. AOP 测试的方法有哪几种？
3. 什么是 SOA？
4. SOA 测试包括哪三个维度？各是什么内容？
5. SOA 测试技术包括哪几个方面？
6. 举例说明如何进行面向对象分析的测试。

第5章 自动化测试

自动化测试是把以人为驱动的测试行为转化为机器执行的一种过程。通常，在设计了测试用例并通过评审之后，由测试人员根据测试用例中描述的规程来一步步执行测试，得到实际结果与期望结果的比较。在此过程中，为了节省人力、时间或硬件资源，提高测试效率，便引入了自动化测试的概念。

本章学习重点

- 了解自动化测试的优点。
- 掌握自动化测试的基本原则。
- 熟悉自动化测试的实现基本策略。
- 了解手工测试和自动化测试的差异。

本章学习难点

熟悉自动化测试的实现基本策略。

5.1 自动化测试的优点

1. 回归测试，降低测试成本

对于产品型的软件或生命周期长的项目，经常会有新功能的开发或需求的变动，对于新发布的软件功能，大部分都和上一个版本相近或相同，这些功能如果在上一个版本之前已经实现了自动化测试，那么新发布的版本中，这部分功能就可以自动化测试实现，避免了重复测试的成本，也确保了软件的质量。

2. 提高测试效率

一些测试用例手工测试是比较烦琐的，例如话单或协议字段的检查，如果是人工检查将是一件既烦琐又耗时还容易出错的工作，如果是自动化测试，测试就会变得轻松和容易很多。

对于检查点很多的测试用例，如果手工执行一步都需要停下来检查好几个复杂的检查

点，测试的效率自然非常低，使用自动化测试，设置好了输入条件和预期结果，只要单击按钮运行一下脚本就知道了复杂的测试结果。

3. 易于发现软件的改动

自动化测试脚本可以重复执行，容易发现软件的任何变动。例如修复了一个 TR 后，引起原功能的改动，执行相同的脚本，可以通过测试轻易发现问题。

4. 充分利用资源

自动化测试可以不需要人在现场的情况下自动执行，发布了一个新版本的软件后，可以在白天的上班时间进行新功能的手工测试，原有功能的自动化测试可以在晚上或周末执行，第二天上班就可以看到执行的结果。这样充分利用时间资源，提高测试的效率，也避免了开发和测试之间的等待。

5. 性能测试

在一些压力大的性能测试中，人工是很难模拟的。在没有引入自动化测试工具之前，为了测试并发，研发中心再加上公司的其他部门上千人在研发经理的口令"1－、2－、3！"的号召下，大家同时单击同一个按钮。这样的测试，虽然是模拟了并发，但需要消耗相当大的成本，想要测试一次也不容易。

在性能测试中使用自动化测试，可以轻易模拟并发，为性能压力测试提供了更好的方法。

6. 将精力投入更有意义的测试

自动化测试减轻了很多重复的工作，测试工程师有更多的时间去思考如何提高软件的质量，制定详细的测试计划，精心设计测试用例，构建更复杂的测试。

自动化测试的好处有很多，但并不意味着自动化测试可以取代手工测试，也不意味着任何的系统都适合自动化测试。自动化测试的意义并不是取代人在测试中的位置，而是将人从重复烦琐的工作中解放出来，做更有价值的测试工作。

5.2 自动化测试基本原则

1. 适合做自动化测试的软件

很多公司都知道自动化测试可以提高测试的效率，但在知道这个道理的公司中大部分都是以手工测试为主的，原因可能有很多，但最大的影响因素就是自动化测试的成本。如果自动化测试的成本比手工测试的成本还大或者并没有比手工测试占有太大的优势，公司自然不会选择自动化测试了。是否做自动化测试，主要看成本的投入和效果的产出是否值得。

1）不适合做自动化测试的系统

（1）系统业务逻辑和交互过于复杂

如果系统业务逻辑和交互过于复杂，要实现自动化测试的成本非常高，工具开发和脚本编写的时间可能远远大于手工测试，这个系统就没有自动化测试的必要。

(2) 项目周期过短

如果系统的生命周期很短(半年内),即使很容易实现自动化测试,但自动化测试的使用率只有很短的时间或很有限的次数,这样的自动化测试也没有必要。因为前期脚本的编写和后期的维护都需要很多的时间,虽然自动化测试在功能测试的过程或回归测试的过程会节省一些时间,但如果自动化测试的脚本只是很短的生命周期,自动化测试的成本就非常高了。

(3) 系统需求频繁变动

对于功能不稳定的系统,会由于这些不稳定因素导致自动化测试失败,自动化的测试结果也就变得不可信,这类型的系统也不适合使用自动化测试。

2) 适合做自动化测试的系统

适合做自动化测试的系统,通常是一些生命周期比较长的项目或产品,且系统功能实现自动化测试也较为容易,这样的项目使用自动化测试必然可以节省很多的资源和成本。特别是一些在今后的几年间需要不断开发和维护的项目,需要重复地进行大量的回归测试,如果有完善的自动化测试脚本,回归测试就可以节省大量的时间和精力了。对于一些增量式的产品,白天手工测试新功能,晚上或周末利用自动化测试脚本回归测试,可以达到资源使用的最优化,用很少的时间和很少的资源做很多的事情。

简而言之,是否值得使用自动化测试,就要看它是否具有自动化测试的特点和高的投资回报率。

2. 开始自动化测试的时机

如果是新的自动化测试工具的开发或研究,最好预留一个比较充裕的时间,时间太赶很难设计出精品。如果想在功能测试阶段使用自动化测试,那么自动化测试架构的设计最好能够与代码实现同步,否则如果等代码实现提交测试之后再做自动化测试工具的开发或研究,在功能测试或回归测试的过程中就被动了很多。

关于在项目的什么阶段开始自动化测试,由项目决定,对于需求相对稳定并且是基于成熟的架构上开发的系统,自动化测试脚本最好在功能测试开始之前编写,在功能测试阶段就可以使用已经编写好的脚本做功能测试了。

但人们平时遇到的项目,有很多是需求变化比较大的,或者是一些不够成熟的系统,这样的系统如果在功能测试之前编写好的脚本,很有可能不能在系统上正确运行,大多还是需要手工执行才可以测试,甚至在功能测试完后的系统跟功能测试之前的系统会有非常大的区别。对于这样的项目,自动化测试开始的越早,项目的成本就越大,最好在系统的架构或需求相对稳定后再做自动化测试。

对于一些需要录制 GUI 界面的功能的自动化测试,在页面的功能相对稳定之后再做自动化测试性价比会比较高,因为页面是最容易变动的部分,而且任何一个控件的修改都会导致自动化工具不能识别控件,导致很多自动化测试脚本会跟着做大量的修改,增加了维护的成本。当然,因为页面变化而引起的脚本的改动的大小,也跟自动化测试的架构和写脚本的功力有密切的关系。

对于一些协议或接口相关的功能测试(例如 XML 或 Socket 接口等),是较为容易实现自动化测试的,封装好底层的协议提供给自动化测试脚本调用,即使是协议会有变化,改动起来还是很简单的,维护的成本相对较低。

总地来说,在软件功能达到相对的稳定,没有严重错误和逻辑错误后开始自动化测试,性价比是比较高的。

3. 自动化测试的覆盖率

自动化测试的覆盖率是很多管理层所关心的,很多项目或产品的自动化测试目标之一就是自动化测试的覆盖率。从管理的角度来说,100%的自动化目标只是一个从理论上可能达到的数值,但是实际上达到100%的自动化的代价是十分昂贵的。自动化测试覆盖率越高,测试脚本的维护成本也就越大。因为所构建版本的需求变化的复杂程度,将花费更多的时间在变更测试用例上以使它们能够正确地运行。

自动化测试的覆盖率的大小与自动化测试的成本有着很大的联系。自动化测试的覆盖率为多少比较恰当,也要看被测试系统的性质和测试的阶段。

在自动化测试设计的阶段,可以考虑先实现冒烟测试的测试用例自动化,冒烟测试的功能一般是系统的主要功能,是自动化测试设计必须首先实现的,而且通过实现这些功能,也可以检验自动化测试的架构是否合理。

在功能测试的前期,自动化测试脚本的覆盖率最好只是一些关键的并且是相对稳定的功能的测试自动化,用于冒烟测试或关键功能测试。

系统稳定后,如果系统是一个生命周期很长的系统,且测试的功能很容易实现自动化测试,这样的系统自动化测试覆盖率可以考虑在80%以上。

但如果是一些时间很赶的项目,或者是一些比较难实现自动化测试的功能,也就没有追求高的自动化测试的覆盖率的必要。随着测试案例的增加,维护的成本也会相应增加,特别是一些GUI的测试,自动化比率越高,维护脚本的成本也就越高。

不要追求在很短的时间实现自动化测试,也不要追求100%的自动化测试覆盖率,积累经验循序渐进的自动化测试,效果会更好。

5.3 自动化测试实现基本策略

自动化测试与软件开发本质上是一样的,利用自动化测试工具,经过测试需求分析,设计出自动化测试用例,从而搭建自动化测试的框架,设计与编写自动化脚本,验证测试脚本的正确性,最终完成自动化测试测试脚本(即主要功能为测试的应用软件)。

1. 测试系统需求分析

任何测试的基础都是被测系统的功能,不管是手工的功能测试还是自动化测试或者是性能测试,都是基于系统的功能展开的。当测试项目满足了自动化的前提条件,并确定在该项目中需要使用自动化测试时,便开始进行自动化测试需求分析。此过程需要确定自动化测试的范围以及相应的测试用例、测试数据,并形成详细的文档,以便于自动化测试框架的建立。

很多公司都是将自动化测试和功能测试划分成两个不同的团队,自动化测试团队的同事实现自动化测试工具的开发,功能测试团队的同事向自动化测试团队的同事提需求,自动化测试团队的同事编写代码实现自动化测试工具的功能后提交给功能测试团队的同事使

用，这是当前非常常见的自动化测试的模式，毕竟每个人都有自己擅长的技能，每个人也不可能面面俱到，通过这样的一种方式可能使自动化测试的门槛变得更低一些。自动化测试工具的开发和自动化测试的使用的确是可以由不同的角色去承担，不过作为自动化架构设计的人员，应该是对系统的功能或需求非常熟悉，同时具有良好的设计和开发能力，才可以设计出适合测试系统的自动化测试架构，否则开发出来的自动化测试工具就只是简单的一个工具，而某种程度上还会增加维护的成本。

漂亮的自动化测试架构的设计是一个渐进的过程，但这个渐进是基于对功能熟悉的基础上，全盘考虑之后一点儿一点儿地搭建起来的。

2. 自动化测试工具的选择

工具始终是工具，思想和架构才是自动化测试的核心，同样的工具不同的人使用会出现完全不一样的结果，而且，不管是什么样的自动化测试工具，原理都有异曲同工之处。所以，在进行自动化测试时，不需要把工具看得那么重要，而应把怎样使用工具，怎样利用工具为测试服务放在首位。也就是测试的思想位于工具之上。

但是，是不是这就意味着测试工具一点儿也不重要呢？当然不是，遇上不稳定或不友好的测试工具，可能会浪费大量的时间在调试工具上，也可能会出现因为工具不稳定导致测试结果的不可信任，那么自动化测试不是提高测试效率反而是阻碍了测试的进度。

关于工具的选择或开发，基本的原则如下。

(1) 能够满足项目的需求，容易扩展，可以满足系统任何重要功能的自动化测试；

(2) 友好易用，容易上手，为测试人员提供较低的门槛；

(3) 当然最重要的是它的稳定性，是否不需要人工干预就能稳定地批量运行所有的自动化测试脚本，并且能够导出准确的测试报告；

(4) 还有一点就是它的性价比是否值得，免费的软件对公司来说当然是最好的。

例如，市场上有很多测试工具，在这些测试工具中，PureTest 是一个性价比很高的自动化测试工具。它容易入门，易于扩展，使用简单，运行稳定，基本上可以满足任何包括 GUI、协议和业务逻辑的测试。

3. 自动化测试架构设计

自动化测试架构的设计是整个自动化测试的灵魂核心，它的好坏关系到自动化测试的成败。从系统的基本功能入手，设计自动测试架构，这是软件测试的关键一步。架构的好与坏很重要，它影响到系统的扩展、维护和使用，如果想要系统容易扩展和维护，一定要多花心思在设计上。用什么测试工具其实并不那么重要，不同的人使用同样的测试工具，会做出效果完全不一样的自动化测试，那是因为他们的测试架构不同，设计比工具重要得多。

怎样的自动化测试架构才算是一个好的架构？第一，容易扩展，能够满足现在的功能需求，也能满足以后需要测试的功能的需求。第二，容易维护，当业务流程、接口或页面变动的时候，只需要做一些简单的修改就可以实现。第三，可读性强，别人能够容易读懂和接手维护。第四，容易使用，项目组的其他人想要使用的时候能够简单地搭建和运行。第五，有统一的编码规范、存储规范和编写风格。第六，方便调试，当脚本运行出现问题的时候，可以方便跟踪问题产生的根源。第七，结构清晰，测试用例与测试工具和代码分离。第八，也是最

重要的一点，是脚本具有很高的可信性以及自动运行的稳定性。

在设计架构之前，首先要熟悉测试系统以及这个测试系统需要测试的功能有哪些类型，每种测试类型在测试架构中是否都可以满足。在设计架构时，可以选择覆盖系统基本功能的冒烟测试用例来做基本的测试用例，在实现这些基本的测试用例的自动化测试过程中，对架构进行完善。基本的自动化测试框架实现之后，再回顾一下是否测试系统中需要实现自动化的测试用例，测试架构都可以满足需求，如果不可以满足则需要继续做进一步的开发，如果测试架构可以满足需求，接下来可以让其他的同事使用，收集改进的建议对测试架构进行完善和改进。好的测试架构，是要使用的人觉得舒服。

自动化测试架构设计的时候的一些基本策略如下。

（1）自动化测试脚本与自动化测试架构的代码分离，自动化测试架构的代码实现自动化测试的基本功能，自动化测试脚本包含业务逻辑。

（2）设计清晰的脚本和配置文件的存放目录。

（3）数据驱动。

① 测试系统相关的配置、模拟器的配置等系统级的配置用系统级配置文件存放，不要把这些值固定地写在脚本或代码中，否则当测试系统的环境变化的时候相应的维护成本也会很高。

② 测试系统所使用到的一些规范定义取值，定义在配置文件中，在脚本中需要使用的时候引用变量定义，这样即使规范定义改变了也不需要修改脚本，只要简单地修改配置文件即可；如果外部规范定义和内部的定义取值不一样，最好有对应的映射定义表。

③ 测试数据的数据模型如果比较复杂，最好也在配置文件中对数据模型以及字段的取值进行定义，方便在脚本中创建数据时使用。

④ 协议或 Schema 或话单的格式，在配置文件中定义，当协议的格式改动的时候，只需要修改配置文件即可。

⑤ 脚本中尽量少用常量，输入参数、脚本运行时提取的值、测试结果的对比等，都可以使用变量，避免脚本的常量写死后引起的维护工作。

（4）测试数据准备。

① 测试数据准备，有两种类型，一类是脚本运行前事先可以准备好的数据，这种类型的数据，可以在自动化测试前自动创建好并保存到文件中提供给测试脚本使用；另一类是脚本运行时要创建的特殊数据，这些数据可以在脚本运行的过程中调用基础脚本进行创建。

② 公用的数据，如果在脚本运行的过程中被修改，在该脚本运行结束后需要恢复到原样，避免因为公用数据的修改引起其他脚本运行失败。

（5）模块化：对基础脚本进行封装，一些可以公用的脚本单独封装给其余的测试脚本调用，当公用的业务逻辑改变的时候，只需要修改基础脚本，而不需要对所有的测试脚本进行改动。

（6）提供自动化脚本编写模板，新写的脚本只需要在模板的基础上做很小的改动就可以实现功能，可以节省脚本编写的时间和降低脚本编写的门槛。

4. 自动化测试脚本编写

自动化测试脚本的编写功力很重要，写得好的脚本，可以减少维护的工作量。自动化测试脚本一般由测试的输入、业务逻辑、测试输出和测试结果验证几部分组成。自动化脚本的编写，要注意全局的把握和审核，不同的人会有不同的风格，稍不注意就会出现问题。在自动化脚本编写前，给相关同事提供技术和架构的培训，培训的内容包括被测试系统的基本功能介绍、自动化测试工具的架构、自动化测试的配置说明、自动化测试的编写原则、自动化脚本编写示例等。自动化测试脚本的例子也很重要，建议在脚本编写前对系统准备实现自动化测试的功能进行分类，由资深的自动化测试工程师根据每个分类都先写一个例子并且审核通过后作为这些功能的自动化测试脚本的编写模板，其余的同事可以参照例子按照自动测试架构编写规范写脚本。

编写脚本时应该注意脚本的可重用性和可维护性，如果脚本中充满了硬编码的值，这些值应该被参数化，否则脚本仅适用于一个测试情况，脚本还应该加入条件判断、循环等结构，以便增强测试脚本的灵活性。

在编写脚本的时候必然会遇到技术问题或业务问题，需要有资深的工程师或团队组长协助解决，并且在整体的架构上全局把握。脚本编写完成后，需要有一个抽查审核的过程，挑几个典型的脚本审核一下，看看是否存在不足的地方，然后改进。

5. 自动化测试脚本测试

当每一个测试用例所形成的脚本通过测试后，并不意味着执行多个甚至所有的测试用例就不会出错。输入数据以及测试环境的改变，都会导致测试结果受到影响甚至失败。而如果只是一个个执行测试用例，也仅能被称作是半自动化测试，这会极大地影响自动化测试的效率，甚至不能满足夜间自动执行的特殊要求。

自动化测试脚本最基本的原则是测试结果可信，也就是在批处理运行这些脚本的时候，该测试通过的就测试通过，该测试失败的就测试失败，如果出现本应该失败的脚本在运行的时候通过了或本应该通过的脚本在运行时失败了，测试结果就变得不可信，自动化测试也就失去它本应该有的意义。

因此，脚本的测试与试运行极为重要，它需要检查多个脚本不能依计划执行的原因，并保证其得到修复。同时它也需要经过多轮的脚本试运行，以保证测试结果的一致性与精确性。

6. 自动化测试脚本执行

自动化脚本主要有三个用途：功能测试、为手工测试做数据准备和回归测试。在功能测试的阶段，可以利用自动化测试脚本进行数据的准备，也可以利用自动化脚本进行功能测试。在项目稳定之后自动化测试的最大价值就是回归测试。

脚本可以分为三个级别：基本流程测试脚本，用于每次出新 build（新构建版本，下同）安装后做冒烟测试；关键功能测试脚本，每次出新 build 后对所有重要功能进行回归测试，确保改动不会对原有功能造成影响；全面回归测试脚本，系统经过比较大的修改或系统上线前做回归测试。自动测试脚本在回归测试中发挥了出色的作用，特别是系统在上线前夕，

为了适应客户的需求,功能不断修改,对于原有的功能,自然不可能都手工测试,脚本在这个时候的意义特别大。

7. 自动化测试的持续集成

自动化测试可以做到持续集成,从编译到测试,任何一步都可以自动化。

(1) 将所有的源代码存放在服务器,持续集成任务运行起来后到源代码管理服务器上进行自动编译,对编译的结果进行分析,并将编译成功的软件版本放到发布服务器;

(2) 将新版本的软件下载到测试环境,并且自动安装;

(3) 自动安装成功后进行冒烟测试,如果冒烟测试成功则证明软件的版本是可用的;

(4) 自动执行自动化测试脚本进行功能测试或回归测试;

(5) 自动化测试结束后生成测试报告,将测试结果发送邮件给相关的人员。

在持续集成中任何一步失败都会导致整个测试失败,自动化测试生成失败的测试报告,并将测试结果发送给相关的人员。

5.4 手工测试和自动化测试的比较

1. 测试项目立项

对任何一个软件开发项目而言,测试本身也是一个工程化的过程,它同样需要经过项目的立项这一个步骤,这一个步骤所要解决的问题主要是解决测试组织的框架的问题,主要是根据测试任务的规模大小来做出相应的先期准备工作;主要是确定项目负责人。测试项目立项对手工测试而言,比较自由,主要是测试项目负责人的问题,以后的测试工作就是由测试项目负责人和整个项目的项目经理来进行沟通完成。对自动测试而言,项目立项中不但要解决前面所提及的问题,还有一个重要的问题是自动测试工具的选择问题。对于自动化测试工具的选择往往要取决于所测试项目的性质。若是测试项目的内容主要是在于测试软件产品的 UI,那么就应该选用 GUI 测试工具如 IBM 公司的 Rational Robot;若是偏重于测试系统性能,可以选用 LoadRunner;若是想实现单元测试的自动化则可以选用 JUnit 等。但是无论选择什么样的工具,一定要给使用人员以专业的培训,这样可以收到事半功倍的效果。

2. 测试计划的编制

测试计划是根据用户需求报告中关于功能要求和性能指标的规格说明书,通过定义相应的测试需求报告,同时,还要适当选择测试内容,合理安排测试人员、测试时间及测试资源等。测试计划活动包含对下面问题的回答。

(1) 什么和在哪里?——要测试什么和在哪里执行这些测试?

(2) 为什么?——为什么要做这种测试?

(3) 什么时候?——什么时候测试必须执行和必须通过?

(4) 谁?——由谁来执行这些测试活动?

测试计划编制的第一步是去确定测试输入。测试输入是测试的依靠或是测试需要的验

证。测试输入帮助用户决定需要测试的内容,它们帮助我们确定基于开发过程中可能需要的变化。在确定了测试输入时,比较容易形成一个测试计划。测试计划为项目中其他的测试提供一个组织结构。需要注意的是,在一个测试项目中可能包含多样的测试计划。可以为测试的每一个阶段编制一个计划。不同的测试工作组可以有他们自己的测试计划。一般地,每一个计划应当有一个确定且唯一的高水平的测试目标。

在软件测试计划编制这一点上,手工测试的重点在于测试进度的安排,测试人员的分配上,以及测试资源的调配上,通常是要生成出软件测试计划说明书。后面的测试活动都是按照测试计划说明书进行逐步展开的。

而对自动化测试则是在上一步选定的测试自动化工具的基础上进行展开的,利用自动化测试工具自身附带的管理工具,或者与自动化测试工具配合使用的测试管理工具,直接在工具中生成实际的测试计划。例如,Rational Test Suite 中的 Rational TestManager 就是一个很好的测试计划生成工具。自动化测试中的资源调配也可以在其中进行分配。但是这些管理工具的缺点是无法应付短时间突发事件,有些缺乏灵活性。

3. 测试的设计

测试的设计主要解决“我们将如何执行测试?”这个问题。一个完整的测试设计会告知我们有关需要与系统被获得的活动和它们应该期待观察的行为和特性,当然,如果系统正在适当地运行的话。

测试的设计是一个迭代的和行进中的过程,应当能够在任何系统执行之前开始测试的设计,它们是基于用例(use cases)、需求、原型和其他的资源而产生的。当系统被描述得更加清晰时,测试的设计应当与系统一起更加细节化。需要注意一个测试的设计不同于软件的设计工作,它应当被用来作为建立测试用例的指导说明书。

手工测试中的测试的设计多是利用了因果图、等价类划分方法、边界值分析方法、错误推测方法、因果图方法、判定表驱动等这些方便人工分析的方法来完成测试用例的设计,使用这些方法来完成测试的设计对测试人员的要求比较高,在正式的测试中,这些工作都是测试项目负责人或者有相当经验的高级测试工程师来完成的。这是因为在这一分析的过程中,更多需要的是工作的经验,人的智慧,人的责任心和耐性。这一过程同样是比较烦琐和关键的,它直接决定了最后得到的测试用例的质量。

而在自动化测试中,测试工具多是使用了类似于软件开发过程中的那种迭代的方法来完成测试的设计的,它是一个逐步求精的过程。而且对测试的分析过程中出现的遗漏的条件很容易在下一次迭代过程中加入。相对手工测试而言,这是自动化测试在测试的设计过程中的最大的优点。例如,在 Rational Test Suite 的 TestManager 中,可以通过下列步骤完成测试的设计。

(1) 指明基本步骤需要与应用和系统交互,以便执行测试。就是测试项目中测试用例在未来的测试脚本中的安排顺序的考虑。

(2) 指明如何有效地使特征恰当地工作。就是解决如何安排测试的校验点的问题。

(3) 说明测试的前置条件和后置条件。解释测试的先导条件和后续的现场回复工作。

(4) 说明测试的可接受标准。说明清楚测试所要达到的精度,尤其是在测试那些与数据处理程序相关的项目中,这一点尤其重要。

所以在自动化测试中，测试的设计比一个手工测试的设计要更抽象，但是它可以容易地发展成一个测试的执行，但是自动化测试方法中测试的设计会受机器配置的制约。如机器系统硬件系统，软件系统的配置，网络系统的配置等，这些在自动化测试过程中必须考虑进去，但是手工测试不一样，它比较灵活，人工可以很容易地改变机器的配置，而后再次进行测试。

4. 测试的实施

测试的实施其实在这里对手工测试而言是要得到真正可用的测试用例，以形成测试执行人员在进行测试执行时的依据。这一步骤对手工测试尤其简单，因为在手工测试过程中一旦完成了测试设计，那么测试用例的生成就是一件极为容易的事情。在大多数的手工测试项目中，测试主管往往是将这一步骤和上面的测试的设计混合在一起来执行的，最终是要形成可视的文档。

对自动测试而言，测试实施的活动包括可复用测试脚本的设计和开发，测试脚本用来实施测试用例。在使用了迭代的方法完成测试的设计后，就可以得到实际上可以使用的测试用例，这时可以将测试用例和根据测试用例开发出来的测试脚本进行结合，将它们联系在一起。在这种联系的过程中，脚本和测试用例之间的关系并不一定是一一对应的，大多数情况下，都是一个测试用例对应一个或者一个以上的测试脚本。

在每一个测试的项目中，测试的实施都是不相同的。需要注意的是，自动化测试中往往要求将测试用例和开发出来的测试脚本组合进行顺序、优先级、执行次数上的安排。如在 Rational Test Suite 的 TestManager 中，它提供了一种称为 Suite 的组合方式，就是将许多的测试用例和脚本进行组合编排，形成一个测试 Suite，在 Suite 中可以添加同步点、时间延迟命令、场景等，使得在一个测试 Suite 中可以在多台不同测试机器上同时执行。这种 Suite 还支持嵌套使用。

相对手工测试而言，自动化测试在这一步骤中要花费比手工测试更多的时间。但是测试脚本的质量的好坏直接影响了整个测试项目的进行。所以这一个步骤对自动化测试项目而言是其关键。

5. 测试的执行

测试执行对手工测试而言，就是一个启动机器和被测试的程序，按照测试用例规定的步骤一步步执行测试的过程。在这个过程中，输入的数据的值及类型均已经在测试用例中有了明确的规定，预期的输出在测试用例中也做出了清楚的说明。测试人员只是被动地执行测试用例过程。这个过程对测试人员而言是一个比较枯燥且乏味的阶段。只是到了测试输出后，测试人员需要将测试的实际输出和预期输出进行人工的比较来确定软件中是否有 Bug 存在。

但是对自动化测试而言，我们的测试执行活动包含要求自动测试的实施的执行就应该确保系统功能的正确性。通过自动化测试工具，可以做到：

(1) 一个单独的测试脚本；

(2) 一个或更多的测试用例；

(3) 一个测试的集合(如 TestManager 中 Suite)，执行一些测试用例和测试脚本的混合

体,这一过程是通过一台或更多的测试用机和虚拟测试者来完成的。

例如,在 Rational TestManager 中就提供了如下几种测试。

(1) GUI 测试——图形界面测试;

(2) VU 测试——虚拟用户测试,多用于性能测试中;

(3) VB 测试——针对 VB 的测试;

(4) Java 测试——针对 Java 系统的测试;

(5) Manual 测试——手动和自动的结合测试;

(6) CommandLine 测试——命令行的测试。

需要注意的是,自动测试一旦启动执行,只要测试脚本是正确的,一般情况下,它的执行是不会轻易被终止的,除非是脚本执行结束或者测试人员人工干预终止。由于这种特性,所以,自动测试大多利用的是测试人员的非工作时间来执行的。这样能大大提高人工效益。但是,相对手工测试而言,自动化测试的执行对脚本的依赖程度极高,一旦测试的对象发生改变或者测试环境发生了改变,就会导致测试脚本出错,使得整个测试无法顺利执行。而手工测试不一样,手工测试最大的特点是灵活性极高。一旦测试的对象或者测试环境发生改变,测试执行人员也能很容易地发现错误,即使是测试用例没有改变。所以在测试执行的灵活程度上,手工测试的程度要好。但是自动化测试的费效比比较高。

6. 测试的评估

测试的评估活动包括:

(1) 确定实际测试执行的有效性。执行得是否完全?执行失败是否因为不符合前置条件?

(2) 分析测试输出以确定结果。在执行测试过程中,查看报告上已产生的数据来检验该执行是否是可接受的。

(3) 查看合计的结果以检查对测试计划,测试输入,配置等的覆盖程度。这也可以被用来衡量测试的进展和对分析的趋向。

(4) 评价本次测试的有效性等。若是有需要还要向上级给出相应的评价报告。

对手工测试而言,这个任务是通过统计过程来完成的。测试人员完成测试后,需要将相关的测试结果记录下来,同时还要统计执行失败的测试用例,从这些数据中形成本次测试报告,上交给测试项目负责人后,由测试项目负责人来完成整个测试项目情况的汇总,并在汇总的基础上完成对测试项目的分析。手工测试中往往要使用到将统计到的大量数据进行公式化的计算才能得出最终的结论和形成最后的报表。所以相对而言比较麻烦,工作量大。

对自动化测试而言,这一切操作起来相对比较简单,一般情况下可以借助自动化测试工具中带有的报告生成机制完成这个任务。在 Rational TestManager 中,它里面的 TestLog 清晰地记录了测试执行,测试过程中发生的问题,测试用例成功和失败的比例,测试过程 Log 信息,同时在这些信息的基础上,运用提供的工具就可以很容易地生成在这个项目中包含的测试对象的情况汇总。但是,需要注意的是,自动化测试工具能对给定的预期的输入(脚本中设定或者取自于测试脚本的数据池)和产生的输出进行比较,但是工具本身并不能告诉测试者软件是通过了测试还是没有通过——它只能说明实际输出结果和与预期结果是否相符。这也是自动化测试和手工测试的一个很重要的区别。所以对于采用自动化测试方

式执行失败的用例往往还是需要使用手工的方式来检查分析，所以从这个程度上来说，手工测试是自动化测试的一个必要的补充。

在上面的几个步骤中，按照测试过程展开的顺序依次比较了手工测试方法和自动化测试方法的区别，可以看出来，手工测试最大的优点是灵活，最大的缺点是费效比太低。但是自动测试最大的优点是它处理得快捷，但是不灵活。下面就结合手工测试和自动化测试各自的特点谈谈它们各自适合的场合。

7. 手工测试和自动化测试的特点及各自适用场合

1）手工测试的特点

（1）测试人员要负责大量文档、报表的制订和整理工作，会变得力不从心。

（2）受软件分发日期、开发成本及人员、资源等诸多方面因素的限制，难以进行全面的测试。

（3）如果修正缺陷所需时间稍长，那么想将手工测试应用于回归测试将变得异常困难。这是因为需要测试的测试用例太多。

（4）对测试过程中发现的大量缺陷缺乏科学、有效的管理手段，责任变得含混不清，没有人能向决策层提供精确的数据以度量当前的工作进度及工作效率。这样往往会导致最后的汇总报表数据不准确。

（5）反复测试带来的倦怠情绪及其他人为因素使得测试标准前后不一，测试花费的时间越长，测试的严格性也就越低。

（6）难以对不可视对象或对象的不可视属性进行测试。

2）自动化测试的特点

（1）可以运行更多更频繁的测试用例。

（2）可以执行一些手工测试困难或者不可能做的测试。如对不可视对象的测试，利用面向对象的自动化测试脚本就很容易实现。

（3）可以更好地利用资源。在夜间执行自动测试用例。

（4）测试具有移植性和可重复性。好的测试脚本往往具有较好的平台移植性。

（5）可以更快地将软件推向市场。因为自动测试节省了大量的时间。

但是自动化测试要求的先期投入比较大，而且要求人员必须经过严格的培训。

所以手工测试和自动化测试各自适用的场合如下。

（1）测试很少执行的项目。当测试用例执行频度太小时（一年一次），可以直接使用手工测试。

（2）软件运行仍然不稳定时，适合使用手工测试。

（3）测试结果很容易通过人验证的测试项目适合手工测试。

（4）测试项目中涉及物理交互比较多的时候适合手工测试。如需要经常查看打印机，绘图仪的输出时。

（5）软件维护时使用的回归测试适合自动化测试。

（6）执行压力测试时适合自动化测试。例如，测试服务器的最大访问上限等。

（7）配置和兼容性测试等项目适合自动化测试。

8. 案例演示和总结

在一个实时的项目监控的系统中，客户通过手机或固定电话拨号完成数据的输入，当接收到的号码一旦与已知设定不符合，触发报警系统，在打印该输入号码同时还要将它转存到磁带上。

测试分析：在该项目中，需要对客户号码、报警器，还有输出设备（打印机和磁带机）这三个方面进行测试。再进一步分析我们知道，对于电话号码而言可能有很多的形式，但是无论如何，它们的值一定是数字组成的，对接收方来说，只有两种情况，收到了合法的数据和收到非法的数据。所以它适合使用程序来模拟输入数据和根据输入判断预期的输出结果，可以使用自动化的方式来实现。对报警器而言，它只有两种状态：报警或不报警。所以同样可以用合法的数据来触发报警和使用非法数据来测试判断其是不是不报警。所以同样可以实现自动化。再看第三个测试对象，输出设备的测试，如前面所述，对于这种物理设备的测试只能使用手工测试。

从上面的这个简单的例子中可以看到，在任何一个项目中，完全的手工测试和自动化测试都是不可取的，可以将那些重复频度大，测试输入容易使用程序来模拟和测试结果可以预期的测试可以使用自动化的方式来实现。对于那些涉及物理交互的测试用例还是使用手工方式为好。总体而言，手工测试和自动化测试是相互补充的。在一个具体的项目中，到底谁占重要的部分，这要根据实际的项目来确定，而不能人为主观限定。同时需要注意的是，在目前的实际应用中大多数的自动化测试所使用的测试用例是从手工测试提取而来的，所以一定要注意对那些手工测试用例的抽象化，否则会导致自动测试脚本中输入数据有遗漏，造成测试输入不完全，这样就会导致测试项目的失败。

总而言之，在现代的软件开发项目的测试中，不存在一个项目完全是使用哪一种方法来独立完成整个项目的测试，这是不可能的。只可能是两种方法同时并重或者一个为主，一个为辅。

思考题

1. 自动化测试有什么优点？
2. 自动化测试的基本原则是什么？
3. 简述自动化测试的实现基本策略。
4. 自动化测试工具如何选择？
5. 请列出手工测试和自动化测试的差异点。
6. Rational TestManager 提供了哪几种测试？

第6章 敏捷测试

敏捷软件开发是目前在业界十分流行的软件开发模式。与传统的软件开发模式有所不同，敏捷开发模式有着自己鲜明的特征。而且，敏捷测试部分也同以往的软件测试流程有所不同。这对测试人员提出了新的要求，带来了新的挑战。本章将结合一个软件项目实例，基于项目开发的不同阶段，介绍每个阶段的主要测试活动。文中将分析每个主要测试活动的前提条件和目标任务，并根据实例推荐最佳的解决方案。

本章学习重点

- 了解敏捷开发的流程。
- 熟悉敏捷测试人员的素质和职责。
- 掌握敏捷开发中的测试流程。

本章学习难点

掌握敏捷开发中的测试流程。

6.1 敏捷软件开发简介

敏捷软件开发(Agile Software Development)起源于20世纪90年代中期。最早是为了与传统的瀑布软件开发模式(Waterfall Model)相比较，所以当时的方法叫作轻量级方法(Lightweight Methods)。20世纪初，该方法的17位倡导者建立了敏捷联盟(Agile Alliance)，并将该软件开发方法命名为敏捷软件开发过程。

敏捷联盟在成立之初就总结了以下4条基本的价值原则。

(1) 人员交流重于过程与工具(Individuals and interactions over processes and tools)。

(2) 软件产品重于长篇大论(Working software over comprehensive documentation)。

(3) 客户协作重于合同谈判(Customer collaboration over contract negotiation)。

(4) 随机应变重于循规蹈矩(Responding to change over following a plan)。

基于这4条原则，敏捷软件开发有着自己独特的流程，如图6-1所示。

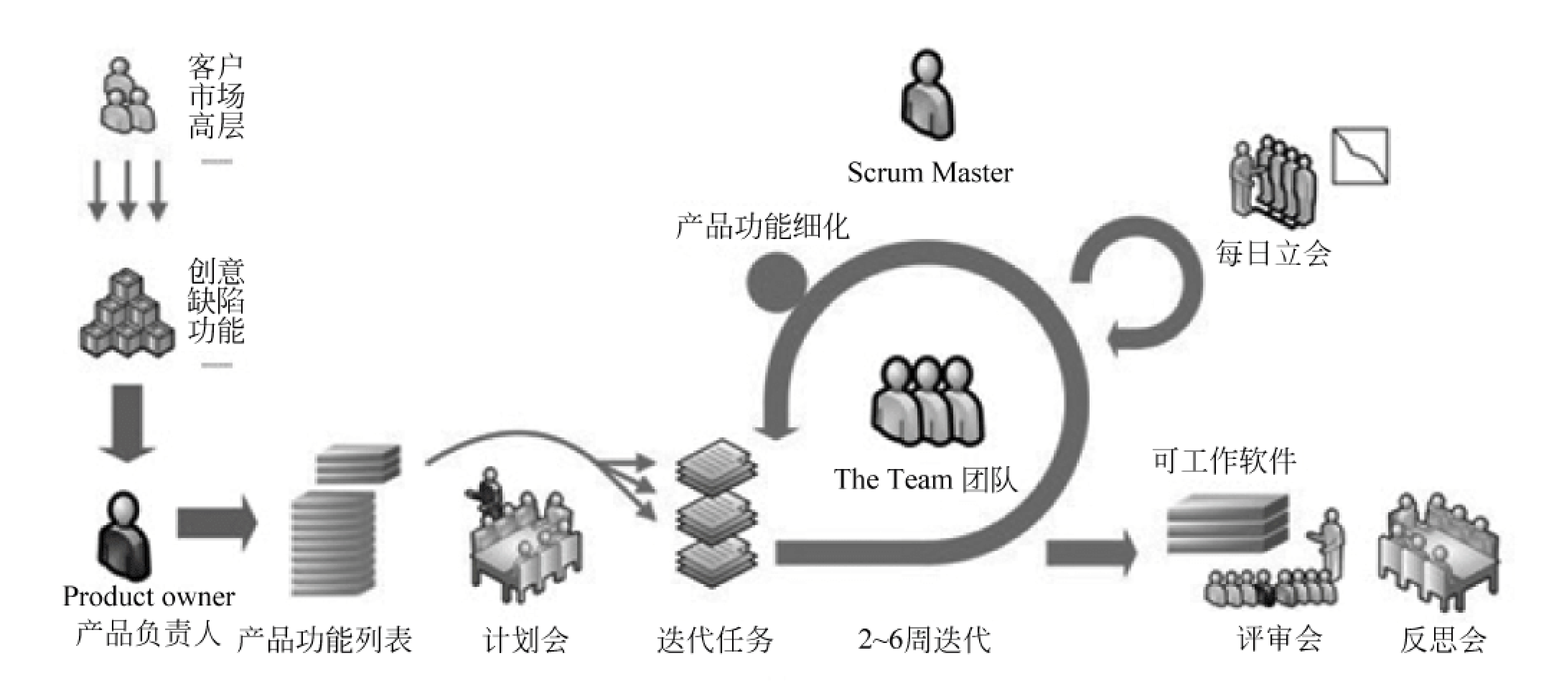


图 6-1 敏捷软件开发流程

整个过程中融合了很多在敏捷开发前已经出现的软件开发方法，包括极限编程(Extreme Programming)、Scrum、特征驱动开发(Feature Driven Development)、测试驱动开发(Test Driven Development)等。这些方法在敏捷软件开发流程的各个阶段都有充分的体现和应用。

例如，Scrum 主要着重于项目管理，团队中的项目经理(Scrum Master)需要在获取到客户需求后制定 Sprint 的周期，定义每个 Sprint 的目标、分派任务、进行监督，最后总结得失并开始计划新的 Sprint。

相反，特征驱动开发和测试驱动开发主要被应用于 Sprint 周期中。如果项目处于开发新功能时期，这个阶段主要推行特征驱动开发。所有开发人员和测试人员都将自己的工作重心放在新的功能上面，从开发和测试两个方面来完成各自的任务。如果项目处于测试新功能时期，这个阶段需要将工作的重点转移到测试上来。所有开发人员和测试人员都密切关注着目前版本的缺陷状况。测试人员需要在每天的站立会议(Daily Standup Meeting)上报告前一个工作日发现的新缺陷情况，项目经理根据项目进度和缺陷严重性来决定是否修复这些问题。需要及时修复的缺陷是目前 Sprint 中的一个新任务，将由项目经理添加到 Sprint Backlog 上并通知开发人员去修复漏洞。

对于敏捷开发和测试中的审查过程，极限编程中的同行评审(Peer Review)思想得到了充分应用。代码和文档的审查追求简单而高效。团队成员两两组成一对，互相评审；有时候，一个开发人员和一个测试人员也可以组成一对，互相协作。这样能够有助于缺陷和问题在第一时间被扼杀在萌芽之中。

敏捷开发还有以下几个关键概念(Key Issues)：迭代过程(Iterative Process)；用户故事(User Stories)；任务(Tasks)；站立会议(Standup Meeting)；持续集成(Continuous Integration)；最简方案(Simplest Solutions)；重构(Re-factoring)。这些概念是敏捷开发中经常使用到的观点和方法。

6.2 敏捷开发中的测试人员

本节将简要介绍敏捷开发中测试人员所需要具备的素质和职责。

1. 敏捷开发团队介绍

例如,敏捷开发团队由4位开发人员、两位测试人员、一位产品设计人员、一位项目经理和一位产品经理组成。每天早上十点,在固定的时间和会议室里面,团队会举行站立会议。这时候,团队成员按照既定的顺序向项目经理汇报各自前一天完成的任务,所遇到的困难和当天要完成的任务。同时,项目经理更新 Sprint Backlog(一张制作精良的 Excel 表格),并及时解决每个人所提出的问题。

由于敏捷开发要求参与人能够快速而高效地应对变化,所以无形中对测试人员提出了很高的要求。

2. 测试人员需要具备的素质

测试是软件开发中不可或缺的一部分。在敏捷软件开发中也是如此。不同的组织给测试人员以不同的称号:测试开发人员(Test Developer)、质量分析员(Quality Analyst)、软件质量工程师(Software Quality Engineer)等。

每个称号隐含不同的职能。以上的称号分别对应以下的能力要求。

(1) 具有质量检测和编写代码的能力→测试开发人员。

(2) 具有防止缺陷(Quality Assurance)和质量控制(Quality Control)的能力→质量分析员。

(3) 具有开发和执行测试程序的能力→软件质量工程师。

总而言之,有三方面的基本素质要求:代码编写(Coding)、测试(Testing)和分析(Analysis)。

在很多其他的开发流程中,各个测试阶段对测试人员的能力有所不同:有时侧重分析(例如系统配置测试),有时侧重代码编写(例如功能测试)。但是,在敏捷开发流程中,测试人员需要结合这三方面来开展工作,只有这样才能真正反映敏捷测试的本质:简单而高效地应对变化。

3. 测试人员的主要职责

在敏捷软件开发中,测试人员的职责有以下三个主要方面。

(1) 定义质量(Define Quality):这应该是软件测试人员的基本职责。敏捷方法鼓励测试人员在 Sprint 计划时直接与客户交流,从自己的经验出发,共同为产品功能制定质量要求。

(2) 交流缺陷(Communication):敏捷过程强调团队中的交流。开发人员经常会专注于重要而新奇的功能,测试人员应该抓住细节,寻找设计中的“missing door”;另外,开发人员使用单元测试来保证产品的基本质量,测试人员可以使用验收测试(Acceptance Test)来鉴定客户需求与实际成果之间的不一致性。

(3) 及时反馈(Feedback):敏捷过程强调简单而高效。测试人员需要及时反馈产品目前的质量问题。这样一来,团队才可以立刻着手解决。如果传统的流程是一周汇总一次状态,敏捷流程要求每天汇总质量问题。例如,在项目中,内部的测试报告以网页的形式显示在内部站点上,每个团队成员能够随时获取;另外,测试框架提供自助测试(Self-assistant Test):通过单击测试用例列表中的某个具体用例,开发人员不需要中断测试人员的工作就可以重现缺陷。

6.3 敏捷开发中的测试流程

本节介绍敏捷测试的最佳实践,详细介绍项目流程中的主要测试活动、每个活动的前提条件和目标任务等。

1. 典型的敏捷开发和测试活动

典型的敏捷开发和测试活动见表 6-1。它主要由三部分构成,从最初的用户故事设计和发布计划,到几次 Sprint 周期的迭代开发和测试,以及最后的产品发布阶段。每个时间段都有相应的测试活动。通常 Sprint 周期被分成两类:特征周期(Feature Sprint)和发布周期(Release Sprint)。特征周期主要涉及新功能的开发和各类测试。发布周期则会结合计划,确定新版本功能,然后对最新的功能进行测试。

表 6-1 敏捷开发的主要活动和测试活动

敏捷开发的主要活动	测 试 活 动
用户故事设计	寻找隐藏的假设
发布计划	设计概要的验收测试用例
迭代 Sprint	估算验收测试时间
编码和单元测试	估算测试框架的搭建
重构	详细设计验收测试用例
集成	编写验收测试用例
执行验收测试	重构验收测试
Sprint 结束	执行验收测试
下一个 Sprint 开始	执行回归测试
发布	发布

在迭代的 Sprint 周期中,开发部分可以根据传统步骤分成编码和单元测试、重构和集成。需要指出的是,重构和集成是敏捷开发的 Sprint 迭代中不可忽视的任务。如果在新的 Sprint 周期中要对上次的功能加以优化和改进,必然离不开重构和集成。

在每个 Sprint 周期结束前,测试团队将提交针对该 Sprint 周期或者上个 Sprint 周期中已完成的功能的验收测试(在实际项目中,测试团队的进度通常会晚于开发团队的进度)。这样一来,开发团队可以通过验收测试来验证所开发的功能目前是否符合预期目标。当然,这个预期也是在迭代中不断变化和完善的。

当产品的所有功能得以实现,测试工作基本结束后,就进入了发布周期。此时,测试团队的任务相对较多。

以上概述了敏捷开发的主要活动。下面将对各阶段相应的测试活动做详细的介绍和分析。首先是用户故事设计和发布阶段。

2. 用户故事设计和发布计划阶段

在用户故事和发布计划阶段,项目经理和产品经理会根据客户的需求,制定概要的产品发布日程计划。此时,测试人员可以和开发人员一起学习新的功能,了解客户的需求。其中,有两个主要活动:寻找隐藏的假设和设计概要的验收测试用例。

1) 寻找隐藏的假设

正如前文所述,开发人员通常关注一些重要的系统功能而忽视细节。此外,敏捷开发倡导简单的实现方案,每个开发 Sprint 周期不可能将功能完美地实现;相反,每个 Sprint 都会增量地开发一些功能。所以,测试人员在最初就需要从各种角度来寻找系统需求,探索隐藏的假设。

2) 设计概要的验收测试用例

定义完一系列用户故事后,测试人员就可以着手设计概要的验收测试用例。正如前文论述,不同于单元测试,验收测试检查系统是否满足客户的预期,也就是用户故事是否能够实现。于是,测试人员可以根据每条用户故事来扩展,寻找其中的"动作",然后为每条"动作"制定正例和反例。

3. 迭代 Sprint 阶段

当一个 Sprint 周期正式开始时,项目经理将制定该周期的具体开发和测试任务。在定期的 Sprint 计划会议(Planning Meeting)上,每位团队成员都要提供自己在未来一个 Sprint 周期中的休假和培训计划。另外,每个团队可以根据各自团队成员的能力和工作经验,适当设定一个工作负载值(Load Factor)。例如,团队的工作负载值为 75%,也就是说每个人平均每天工作 6h(以 8h 计算)。接着就可以开始分配任务。

当开发团队开始编码和单元测试时,测试人员的工作重点包括估算验收测试的时间、估算测试框架的搭建、详细设计验收测试和编写验收测试代码。前两个主要活动一般在项目初期的 Sprint 周期中完成。其他主要活动将在接下来的多个 Sprint 周期中视情况迭代进行。下面将具体介绍每个主要活动。

1) 估算验收测试时间

在软件开发初期,需要估算时间以制定计划。这一点在敏捷开发中应用更加广泛。如果以前的开发模式需要测试人员估算一个软件版本发行的计划(这样的计划通常会延续几个月),那么现在则要在每个 Sprint 机会会议上估算两周到一个月的任务。此外,在每天的站立会议上,测试人员需要不断地更新自己的估算时间,以应对变化的需求。所以,每个测试人员都应该具备一定的估算任务能力。下面将介绍两个通用的估算测试计划的方法。

(1) 快速而粗糙的方法

从经验而言,测试通常占项目开发的三分之一时间。如果一个项目开发估计要 30 人天,那么测试时间为 10 人天。

(2) 细致而周到的方法

这个方法从测试任务的基本步骤出发,进行详细分类。其中包括:

① 测试的准备(设计测试用例、准备测试数据、编写自动测试代码并完善代码);

② 测试的运行(建立环境、执行测试、分析和汇报结果);

③ 特殊的考虑。

2) 估算测试框架的搭建

测试框架是自动测试必不可少的一部分工作。由于敏捷开发流程倡导快速而高效地完成任务,这就要求一定的自动测试率。一个完善的测试框架可以大大提高测试效率,及时反馈产品的质量。

在敏捷开发流程中,在第一个 Sprint 周期里,需要增加一项建立测试框架的任务。在随后的迭代过程中,只有当测试框架需要大幅度调整时,测试团队才需要考虑将其单独作为任务,否则可以不用作为主要任务罗列出来。

3) 详细设计验收测试用例

完成对测试任务的估算,接着就可以着手详细设计验收测试用例。可以对概要设计中的测试用例进行细化,根据不同的测试环境、测试数据以及测试结果,编写更详细的测试用例。另外,可以结合几个用例,完成一个复杂的测试操作。

由于敏捷开发的流程是不断迭代的过程,所以很多复杂的功能可能会在未来的 Sprint 周期中被优化。对测试人员而言,一个有效的方法是尽量将一些验证基本功能的测试用例作为基本验证测试用例(Basic Verification Test Case)在第一时间实现自动化;而对一些复杂的功能测试用例,可以先采用手工的方法测试,直到在未来 Sprint 周期中该功能达到稳定时再考虑自动化。此外,对测试中出现的缺陷可以设计回归测试用例(Regression Test Case),为其编写自动测试代码,使得此类问题在发布周期(Release Sprint)时可以顺利而高效地进行验证。

4) 编写验收测试用例

敏捷开发不提倡撰写太多的文档,提倡直接编写测试用例。此外,测试人员和客户应取得良好的沟通,将需求总结下来,转化成验收测试用例。如果资源充足,最好对验收测试用例建立版本控制机制。

考虑到需求在每一轮 Sprint 周期中会不断地变化,测试团队要控制测试的自动化率,正确估计未来功能的增减。自动化率过高会导致后期大量测试代码需要重构,反而增加很多工作量。

4. Sprint 结束和下一个 Sprint 开始

在一个 Sprint 周期结束时,团队要举行一个回顾会议(Retrospective Meeting)。团队成员可以在会议上畅所欲言,指出在过去一个 Sprint 周期中可行的、不可行的和有待改进的地方。待改进之处将在项目经理监督下于未来的 Sprint 周期中实现。

由于敏捷开发倡导增量开发,当新的 Sprint 开始时,测试团队需要根据新 Sprint 周期的开发进度及时重构验收测试。如果新 Sprint 周期没有具体的新功能开发,测试团队可以将精力集中在执行验收测试和寻找缺陷上。

如果下一个 Sprint 周期是发布周期,那么测试人员需要准备执行回归测试。下面来详细了解每个测试活动。

1）重构验收测试

正如上文所提及，敏捷开发是以迭代方式进行的，功能在每次迭代中推陈出新。于是，验收测试用例经常需要修改或者添加，相应的验收测试代码也需要删减。这部分工作如果时间花销很大，最好在估算的时候一并提出。

2）执行验收测试

验收测试可以分为两大类：基本验证测试和功能测试。如果是基本验证测试，推荐开发人员在运行完单元测试和提交代码前直接运行自动测试脚本。如果是功能测试，可以在每个 Sprint 后期，新功能代码提交后，由测试人员单独执行。

敏捷开发的开发和测试是相辅相成的。一旦基本验证测试出现问题，那就说明开发人员的实现违反了最初客户定义的需求，所以不能够提交。如果功能测试出现问题，那么测试人员就需要及时与开发人员沟通。如果是缺陷，需及时上报给项目经理，并在每天站立会议中提出；如果不是，那么继续下一项任务。这个过程充分体现了敏捷开发所提倡的团队交流机制。

3）执行回归测试

在发布周期中，测试人员所肩负的任务非常重要，因为这是产品发布前的最后质量检验。

首先，要建立一套自动生成 build、运行自动测试代码、手工执行测试用例并汇总测试结果的框架。估算方法参见上文。

其次，定期执行各类测试，包括功能测试和系统测试。

最后，要整理之前在每个特征测试周期中出现的问题。如果已经整理并归类为回归测试用例，那么只要定时执行就可以了；否则，就需要一一添加。如果用例已经被自动化，可以直接运行；如果是手工测试，测试人员需要按照测试用例进行操作，最后汇总测试结果。这部分测试就是所谓的回归测试。

5. 手工测试和测试报告

手工测试和自动测试是两个主要的测试类型。考虑到敏捷开发的高效性，自动测试会优于手工测试。手工测试有两个主要的缺点：不可靠和容易被遗忘。敏捷测试主张一些基本的验收测试可以被自动化；对一些涉及系统方面的测试，手工测试比较适合。

测试报告是反映一个测试团队工作的最好成果。为适应敏捷开发的节奏，测试报告可以以网页的形式发布在内部的 Web 服务器上，在一些问题区域上标注鲜明的色彩，用来警示团队中的每个人。

6.4 案例分析

本节将结合一个软件项目实例来加深对敏捷测试的了解。

1. 项目实例介绍

根据一家在线 B2B 公司的要求，将为其开发一款类似于百度的搜索服务。作为 Web Service，该服务可以内嵌于网页之中。当用户输入关键词并选择商户的类型和位置后，系

统会返回具体商户的列表。

2. 用户故事设计和发布计划阶段

1）寻找隐藏的假设

（1）从在线 B2B 公司角度思考

问：这个搜索框对公司的业务有什么价值？

答：搜索框可以为用户方便地提供商户的目录信息。如果越来越多的用户使用这个搜索框，可以增加网站的访问量。

（2）从用户角度思考

问：作为查询信息、寻找商业合作伙伴的网站用户，搜索框对我有什么好处？

答：坏处：找到一家商户的地址，过去才发现已经关门歇业。好处：查找商户很简单，只要单击鼠标。不快：有时候在寻找一类商户，却记不清楚具体名字。

（3）从程序员角度思考

问：一个搜索框的最简单实现方法是什么？

答：一个由 text input 和 search button 组成的 form；后台通过 server 程序将符合类型和地址的商户名从数据库中取出，返回给用户；每个返回项包括商户的名称、地址和评价意见。

（4）寻找这些观点中的问题

问：搜索框如何在用户忘记具体名字的时候提醒用户？

答：在第一版本中实现比较困难。可以让用户输入至少一个类型来提高模糊查找的效果。

（5）最后寻找到隐藏的假设

以上的思考让测试人员对系统的隐含假设更加清晰：首先，系统应该能够在高峰时候处理 300 条搜索请求和 2000 个鼠标单击事件。其次，用户可以在已经查找到的内容中继续查找。最后，系统提供一个商户类别清单；如果用户选择商户类别而忘记具体名字，系统提供模糊查询。

在敏捷开发中，这些假设可以作为用户故事记录下来，从而指导未来系统的开发和测试。

2）设计概要的验收测试用例

设计概要的验收测试用例如表 6-2 所示。

表 6-2 设计概要的验收测试用例

动作	数　据	期待的结果
搜索	一组能成功搜索到的(类别，位置)数据	在该类别和位置条件下的一组商户信息
搜索	一组不能成功搜索到的(类别，位置)数据	空列表

3. 迭代 Sprint 阶段

1）估算验收测试时间

搜索框的开发估计需要 78 人天完成。但是，考虑到系统有模糊搜索的功能，所以测试任务可能会占 40%左右，大概 31 人天。表 6-3 列出了具体的任务和估计时间。

表 6-3 具体任务及估计时间

任　　务	估计时间
设计测试用例,准备测试数据(搜索数据集)	8
加载数据集	2
编写自动测试代码	18
执行测试和汇报结果	3
总结	31

估算单个测试任务的事例见表 6-4。

表 6-4 估算单个测试任务的事例

测试	准备		运行		特殊考虑	估算
1	设计测试用例	0.5	建立环境	0.1		
	准备测试数据	0.5	执行测试	0.1		
	编写自动测试代码	0.5	分析结果	0.1		
	完善自动测试代码	2.5	汇报结果	0.1		
合计		4		0.4	0	4.4

估算多个测试任务的汇总见表 6-5。

表 6-5 估算多个测试任务

测试任务 编号	准备	运行	特殊考虑	估算
1	4	0.4	0	4.4
2	4	0.4	0	4.4
3	12	4.5	8.5	25
4	4	0.4	0	4.4
5	4	0.4	0	4.4
6	4	0.4	0	4.4
7	4	0.4	0	4.4
合计				51.4

2）估算测试框架的搭建

考虑该项目刚刚进入测试,需要为此建立一个测试框架,见表 6-6。于是,在原先的估算中多增加一些任务。

表 6-6 测试框架

任　　务	估算/h
选择测试工具	3
建立测试系统	3
编写下载、存放和恢复测试数据的脚本	2
寻找或建立测试结果汇报工具	8
设计具体的搜索测试用例	4

续表

任务	估算/h
准备搜索测试数据	4
编写和测试“搜索”模块	3
编写和测试“验证返回列表”的模块	1
学习“在结果中搜索”的模块设计	4
编写和测试“在结果中搜索”模块	4
第一次执行测试	4
分析第一轮测试结果	4
第二次执行测试	4
分析第二轮测试结果	4
合计	52

3）详细设计验收测试用例

基本验证测试用例如表 6-7 所示。

表 6-7 基本验证测试用例

动作	数据	期待的结果
登录	用户名：（空） 密码：（空）	“用户名和密码无效”

功能测试用例如表 6-8 所示。

表 6-8 功能测试用例

动作	数据	期待的结果
登录	正确的用户名和密码	进入系统：请输入搜索条件并单击“搜索”按钮
搜索	错误的类型	提示正确的类型
搜索	使用正确的类型	商户列表

4）编写验收测试用例

测试人员和客户取得良好的沟通后，将需求总结下来，直接转化成验收测试用例，并对验收测试用例建立版本控制机制。此外，还要控制测试的自动化率。

4. Sprint 结束和下一个 Sprint 开始

在一个 Sprint 周期结束时，测试团队举行会议进行总结。找出在过去一个 Sprint 周期中可行的、不可行的和有待改进的地方。改进之处在项目经理监督下于未来的 Sprint 周期中实现。

1）重构验收测试

修改或者添加验收测试用例以及相应的验收测试代码。

在下一个 Sprint 周期中，需要实现之前没有实现的“模糊查找”功能。测试人员要在新的 Sprint 周期中更新原来的验收测试用例，在测试“搜索”模块中添加模糊查找测试。重新估算的测试任务见表 6-9。

表 6-9 重新估算的测试任务

任 务	估计时间
设计测试用例,准备测试数据(模糊搜索数据集)	2
加载数据集	1
编写自动测试代码	3
执行测试和汇报结果	2
合计	8

2) 执行验收测试

开发人员在运行完单元测试和提交代码前直接运行自动测试脚本,进行基本验证测试。在每个 Sprint 后期,新功能代码提交后,由测试人员单独执行功能测试。

如果功能测试出现问题,那么测试人员就需要及时与开发人员沟通。如果是缺陷,需及时上报给项目经理,并在每天站立会议中提出;如果不是,那么继续下一项任务。

3) 执行回归测试

在发布周期中,测试人员需要进行回归测试。

5. 手工测试和测试报告

在该搜索实例中,一旦重新建立索引,那么先前在搜索文本中出现的文字错误就无法重现。另外,当测试人员按部就班地手工完成一个一个测试用例时,他们很容易遗忘一些特殊的测试用例,很多缺陷因此而被埋没。因此一些基本的验收测试可以被自动化;对一些涉及系统方面的测试,手工测试比较适合。

测试报告以网页的形式发布在内部的 Web 服务器上,在一些问题区域上标注鲜明的色彩,用来警示团队中的每个人。

思考题

1. 什么是敏捷软件开发?
2. 敏捷开发融合了哪几种软件开发方法?
3. 敏捷联盟在成立之初总结的 4 条基本价值原则是什么?
4. 敏捷开发中测试人员所需要具备的素质和职责是什么?
5. 举例说明项目流程中的主要测试活动有哪些。

工　具　篇

本篇内容：

- 工具总论
- 黑盒测试工具与白盒测试工具
- 性能测试工具与安全测试工具
- 测试管理工具

第7章 工具总论

本章学习重点

- 了解工具角度的分类。
- 掌握常见的测试工具的对比情况。
- 熟悉测试工具的选择方法。

本章学习难点

掌握常见的测试工具的对比情况。

7.1 工具角度分类

目前用于测试的工具较多，从测试应用的角度看，一般可分为白盒测试工具(动态测试、静态测试)、黑盒测试工具(功能测试、性能测试)、测试管理工具(测试流程管理、缺陷跟踪管理、测试用例管理)等几大类。

1. 白盒测试工具

白盒测试主要是从程序的内部结构出发设计测试用例，对程序所有逻辑路径进行测试，通过在不同点检查程序的状态，确定实际的状态，来测试产品内部动作是否按照设计规格说明书的规定正常进行，检验程序中的每条通路是否都能按预定要求正确工作。其对应的测试工具也主要是直接对代码进行分析，针对程序代码、程序结构、对象、类层次等进行测试，测试中发现的缺陷可以定位到代码行、具体的某个变量。软件自动化测试中对白盒测试工具的选择主要应依据该工具对开发语言的支持力度、对嵌入式操作系统的支持力度、代码的覆盖深度及测试的可视化。

白盒测试工具可进一步细分为静态测试工具和动态测试工具。静态测试工具是不运行被测程序本身，仅通过分析或检查源程序的语法、结构、过程、接口等来检查程序的正确性。具有代表性的静态测试工具有 Gimpel 公司的 PC-Lint 和 Compuare 的 DevPartner Studio 中的 CodeReview。动态测试工具需要实际运行被测系统，并设置断点，向代码生成的可执

行文件插入一些监测代码,监测断点这一时刻程序运行的数据。具有代表性的动态测试工具有 IBMRational 公司的 Purify,Pure Coverage,Quanlify 和 Compuare 公司的 Error Detect,Coverage Analysis,PerformanceAnalysis。

2. 黑盒测试工具

黑盒测试是在已知产品所应具有的功能的情况下,通过测试来检测每个功能能否正常使用的测试工具。其基本工作原理是利用脚本的录制和回放,模拟用户的操作,然后将被测系统的输出记录下来同预先给定的标准结果比较。测试时完全不考虑程序内部结构和内部特性,它只检查程序功能是否按照需求规格说明书的规定正常使用,主要用于软件确认测试。黑盒测试工具的代表有 IBM Rational 的 TeamTest、Robot, Compuware 公司的 QA-Center,MI 公司的 QTP,WinRunner 等工具。

3. 测试管理工具

测试管理工具是指用工具对软件的整个测试输入、执行过程和测试结果进行管理的过程。测试管理工具通过一个中央数据仓库,实现测试人员、开发人员或其他 IT 人员在异地进行信息交流。从测试需求管理到测试计划、测试日程安排、测试执行到出错后的错误跟踪,实现了全过程的自动化管理,提高回归测试的效率,大幅提升测试时间、测试质量、用例复用、需求覆盖等。测试管理工具的代表有 Mercury Interactive 公司的 TestDirector、IBM Rational 公司的 ClearQuest。

7.2 常见的测试工具对比

1. 白盒测试工具

白盒测试工具一般是针对代码进行测试,测试中发现的缺陷可以定位到代码级,根据测试工具原理的不同,可分为静态测试工具和动态测试工具。

1) 静态测试工具

静态测试工具直接对代码进行分析,不需要运行代码,也不需要对代码编译链接,生成可执行文件。静态测试工具一般是对代码进行语法扫描,找出不符合编码规范的地方,根据某种质量模型评价代码的质量,生成系统的调用关系图等。静态测试工具的代表有 Telelogic 公司的 Logiscope 软件、PR 公司的 PRQA 软件等。

(1) (Telelogic)Logiscope

功能:Logiscope 分为 Audit、RuleChecker 和 TestChecker 三部分,对代码分别进行静态度量、编程风格检测和测试覆盖率分析。下面分别就这三部分功能进行描述。

① Audit:以 ISO 9126【3】模型作为质量评价模型的基础。质量评价模型描述了从 Halstead、McCabe 的度量方法学和 Verilog 引入的质量方法学中的质量因素(可维护性、可重用性等)和质量准则(可测试性、可读性等)。

② RuleChecker:使用所选规则对源代码一一进行验证,指出所有不符合编程规则的代码,并对应所违反的规则。

③ TestChecker：测试覆盖率分析工具，提供：指令覆盖、判定覆盖、MC/DC(条件组合覆盖)和基于应用级的PPP覆盖。分析这些覆盖率信息可以保证测试，提高测试效率，协助进行进一步的测试。

同时，Logiscope支持对嵌入式系统的覆盖率分析。首先对应用源代码插装，然后实时地将测试信息通过网线/串口传到宿主机(Host)上，并在线显示。Logiscope支持VxWorks、pSOS、VRTX等实时操作系统。

支持语言：

C，C++，Ada，Java。

是否免费：否

(2)(PR)PRQA

功能：PRQA的主要产品包括：QAC/QAC++，QA. MISRA C/QA. MISRA C++。QAC/QAC++是用于代码规则检查的自动化工具。

主要功能：支持C，C++，Java和FORTRAN应用的静态分析；确定并增强代码的标准化(工业级以及自定义)；通过代码审查度量软件质量；软件编程规则评估；MISRA C准则检查。

① 代码规则检查自动化

这是一个基于C语言开发环境下用以提高软件产品产量和质量标准的深层次静态分析工具软件。这个软件可以自动识别C语言源代码中出现的问题。这些问题主要是语言使用过程不安全，过于复杂，无法移植，难以维护或与该行业的代码标准偏离造成的。QAC能够对许多编译器或其他工具开发软件无法说明的问题提出警告。这个工具将极大地缩减代码检测的时间并能同时加强程序设计人员对C语言中不完全为人理解的某些特点的认知。利用QAC，可在开发软件早期阶段对存在的问题加以注意，如能将代码质量提高，同时测试周期也将缩短。

② 提供深层次的静态分析

使用该工具不但可迅速而有效地检测出语言运用中的错误、已过时用法、程序标准一致性问题，从而防止在软件开发的后期以更昂贵的代价去解决问题，而且还将工业标准分析度量标准和通俗易懂的报告结合在一起。

③ 规则可以定制

对于工具的数据库中已有的规则，可以由测试人员决定使用哪项规则或不使用哪项规则，或者是某一个错误等级的规则。

对于特殊行业来讲，也许部分需求有点儿特殊，在工具中没有所要求的规则，但这没有关系，因为规则是可以人为扩充的，用QAC提供的定制的方法可以添加要的规则。

④ 可以和开发工具集成

可以和Visual Studio 6.0、Visual Studio .NET、Tornado集成，在开发环境中来使用QAC，提高测试效率。

支持平台：Solaris、HPUX、Linux、Windows

优点：

① 缩减软件开发的成本和产品上市的时间。

② 降低软件产品质量问题。

③ 实现代码检测过程自动化，使软件开发和质量检验技术人员提高效率。

④ 在软件研发的早期阶段识别潜在的软件产品问题和其他可能出错的问题，从而减少产品测试和顾客使用中发现问题的几率。

⑤ 具有较好的集成性，能在现有的软件环境下实现安装和卸载。自动检测软件产品是否符合某公司或某行业的软件标准和语言安全性方面的要求。

⑥ 提高C语言代码的编写质量，通过加强软件可靠性、移植性和可维护性三方面来减少软件产品未来的维护费用。

⑦ 帮助软件开发人员生产高质量的代码。

⑧ 支持软件认证，软件研发过程认证和各种质量认证，如CMM认证，ISO 9003/EN29003，TickIT，IEC 61508，Def Stan 00-55，DO-178B。

⑨ 设立了软件质量度量标准，后期代码修改可以得到衡量和比较。

⑩ 为软件开发的成本和产量提供依据。

⑪ 帮助企业培训软件研究及开发人员，使其在利用C语言编程过程中避免问题。

缺点：

① 代码规则检查需要付出很繁重的劳动——重新理解代码，国内一些软件工程发展到现在，已经有了专职的测试人员，即使非常专业的测试人员，理解别人写的代码也是一项很烦琐的工作。

② 时间和资源的限制。任何一个企业都可以做出优秀的软件，前提是给他足够的时间和物质资源，可现实的软件开发的矛盾却是：在有限的时间内、利用有限的经费，来做高可靠性的软件。

③ 很多人不重视代码规则检查，包括很多软件企业的领导、项目负责人等，认为代码规则检查浪费人力和物力，恰恰相反，这种观点就把软件中存在的问题留到了最后，在软件维护过程中会付出昂贵的代价。经验表明，软件中的问题发现得越早，要克服这个问题付出的代价越小。

是否免费：否

2）动态测试工具

动态测试工具一般采用“插桩”的方式，向代码生成的可执行文件中插入一些监测代码，用来统计程序运行时的数据。其与静态测试工具最大的不同就是动态测试工具要求被测系统实际运行。动态测试工具的代表有Rational公司的Purify系列、Compuware公司的DevPartner等。

（1）（Rational）Purify

功能描述：自动化测试工具Rational Purify是Rational PurifyPlus工具中的一种，RationalPurifyPlus包括三种独立的工具：Rational Purify、Rational Purecoverage、Rational Quantify。

Purify面向VC，VB或者Java开发的测试Visual C/C++和Java代码中与内存有关的错误，确保整个应用程序的质量和可靠性。在查找典型的Visual C/C++程序中的传统内存访问错误，以及Java代码中与垃圾内存收集相关的错误方面，Rational Purify可以大显身手。Rational Robot的回归测试与Rational Purify结合使用完成可靠性测试。Java程序员和测试人员可以将Rational Purify和所支持的JVM相结合，以改善和优化Java Applet及

应用程序的内存功效。Rational Purify 可以运行 Java Applet，类文件或 JAR 文件，支持 JVM 阅读器或 Microsoft Internet Explorer 等容器程序。

使用 Rational Purify 特有的 PowerCheck 功能，可以按模块逐个调整所需的检查级别。这样就可以把精力集中在最重要的代码上。简单选择“最小”或“准确”即可。

对于同时进行代码覆盖分析的情况，请选择覆盖级别，如“代码行”或“函数”，以便更好地控制错误检查和数据覆盖。Purify 通过对 API 调用的验证，确保应用程序的可靠性。

Purify 带有及时调试功能，当检测到错误时，它将自动停止编程并启动调试器。也可以通过 Purify 工具栏，将该调试器附加到正在运行的流程中。这将大大增强诊断应用程序中问题的能力，从而缩短查找、复审和修正错误所需的时间。

Rational Purify 可以从多个侧面反映应用程序的质量——功能、可靠性和性能。Rational Purify 通过检测影响可靠性的内存相关编程错误，提高 Java 和 C++软件的质量。Purify 可在进行功能测试的同时，对可靠性问题进行检测，从而弥补了质量测试的不足。

功能描述：

可检查的错误类型：

① 矩阵相关错误。

② 堆栈相关错误。

③ 垃圾内存收集——Java 代码中相关的内存管理问题。

④ COM 相关错误。

⑤ 指针错误。

⑥ 内存使用错误。

⑦ Windows API 相关错误。

⑧ Windows API 函数参数错误和返回值错误。

⑨ 句柄错误。

可检测错误的代码：

① ActiveX(OLE/OCX)控件。

② COM 对象。

③ ODBC 构件。

④ Java 构件、Applet、类文件、JAR 文件。

⑤ Visual C/C++源代码。

⑥ Visual Basic 应用程序内嵌的 Visual C/C++构件。

⑦ 第三方和系统 DLL。

⑧ 支持 COM 调用的应用程序中的所有 Visual C/C++构件。

优点：

① 对 Debug 程序能很好地查出内存泄漏，并且精确定位代码行。

② 使用很简单，在 Rational Purify 环境中运行要调试的程序，Purify 首先会自动插装程序，查找每个内存操作相关指令，并写入加上自己的检测指令，然后将修改完的程序复制到 Purify Cache 目录下运行。在运行中维护一张内存使用表，检测每块内存的分配和销毁，直至最终程序退出，Purify 根据其维护的内存使用表，总结所有没有被释放的内存。

③ Purify 会插装所有程序用到的 dll 执行文件，包括系统 dll，可以设置每个 dll 的插装

程度。

④ 可以使用 Filter 过滤掉一些良性的内存错误。

缺点：

① 基本上对 Release 版本的程序没什么用，一个 Hello World 程序 Release 版都能被它测出 64KB 的内存泄漏，令人费解。

② 极不稳定，崩溃无数，作为一个测试工具自身如此多的致命问题，Rational 公司亟待改进。

③ 文档较差，大部分查到的文档，讲的都是功能概括，而非具体操作指导，实际上解决不了什么问题。

④ 被它插装的大型程序(Release 1MB 以上，Debug 4MB 以上)极不稳定，容易崩溃，运行速度会慢两个数量级，导致根本无法正常运行到进程退出，内存错误查找更无法进行。

是否免费：否

(2) (Compuware)DevPartner

功能：DevPartner Studio 主要支持三个功能，BoundsChecker，TrueCoverger 和 TrueTime。BoundsChecker 是用来检查内存泄漏的，TrueCoverger 是用来检查单元测试代码概率的，TrueTime 是用来检查动态运行时哪段代码的执行效率偏低。

DevPartner Studio 9.0 可以实现的功能包括，扫描基于微软 ASP.NET 的应用程序源码，以找出潜在的应用程序 Bug 和可疑行为。该产品可高亮显示存在问题的代码，这意味着 IT 经理可以在开发早期发现代码问题，防止小 Bug 成为顽疾。另外，此程序还可生成基于 Web 的报告，报告内容包括代码质量、代码审查、错误检测等。使 IT 经理们可以准确地找到问题，构建更加稳定的代码。

支持平台、技术和操作系统：DevPartner Studio 9.0 现在支持 Windows x64 位平台的 32 位应用程序开发，以及最新的.NET Framework 技术，支持的编程工具及操作系统包括 Visual Studio 2008、Windows Server 2008、.NET Framework 3.5 和 Windows Presentation Foundation。

该工具还将支持微软的早期编程工具，例如 Visual Studio 6.0、Visual Basic 6.0、Visual C++ 6.0、Visual Studio .NET 2003 和.NET Framework 1.1。

优点：

DevPartner Studio Professional Edition 能加速软件开发并构建可靠的软件代码，DevPartner Studio 能方便地集成到团队的现有开发流程和 Visual Studio .NET IDE 中，从而对.NET 开发提供了出色的解决方案；它还提供针对.NET 和本地开发的解决方案，帮助程序的维护和优化人员开发.NET 程序。

DevPartner Studio 为软件开发人员对复杂的程序错误提出了最佳的标志方法，帮助他们构建应用程序的各种商务逻辑。

DevPartner 源代码分析器能方便地指出程序错误，更重要的是它能说明程序错误的原因，以指针形式指明网络或书中才有的详细信息。

DevPartner Studio 代码分析器，能够让用户一边编码一边学习，并通过每次发布新版本，提高工作效率。

是否免费：否

2. 黑盒测试工具

黑盒测试工具适用于黑盒测试的场合，黑盒测试工具包括功能测试工具和性能测试工具。黑盒测试工具的一般原理是利用脚本的录制(Record)/回放(Playback)，模拟用户的操作，然后将被测系统的输出记录下来同预先给定的标准结果比较。黑盒测试工具可以大大减轻黑盒测试的工作量，在迭代开发的过程中，能够很好地进行回归测试。黑盒测试工具的代表有：Rational 公司的 Robot、TeamTest；Compuware 公司的 QACenter。

1）(Rational)Robot

功能：Rational Robot 是 Rational 的产品之一，提供了软件测试的功能，行如其名，它提供了许多类似机器人的重复过程，供测试用。

Rational Robot 可开发三种测试脚本：用于功能测试的 GUI 脚本、用于性能测试的 VU 以及 VB 脚本。

Rational Robot 可以让测试人员对.NET、Java、Web 和其他基于 GUI 的应用程序进行自动的功能性回归测试，是一种对环境的多功能的、回归和配置测试工具。在该环境中，可以使用一种以上的 IDE 和(或)编程语言开发应用程序。可以很容易地使手动测试小组转变到自动测试上来。使用 IBM Rational Robot 进行回归测试是早期步入自动化的很好的一步，因为它易于使用，并且可以帮助测试者在工作的过程中学习一些自动处理的知识。

为诸如菜单、列表和位图这些通用的对象提供测试用例和为特定于开发环境的对象提供专用的测试用例。

包括内置的测试管理，并且在 IBM Rational Team Unifying Platform 中整合了错误跟踪的工具，这改变了管理和需求跟踪能力。

支持的技术：支持从 Java™ 和 Web 到所有 VS.NET 控件的多种 UI 技术，包括 VB.NET、J＃、C＃和 Managed C++。

是否免费：否

2）(Compuware)QACenter

功能：QACenter 帮助所有的测试人员创建一个快速、可重用的测试过程。这些测试工具自动帮助管理测试过程，快速分析和调试程序，包括针对回归、强度、单元、并发、集成、移植、容量和负载建立测试用例，自动执行测试和产生文档结果。QACenter 主要包括以下几个模块。QARun：应用的功能测试工具。QALoad：强负载下应用的性能测试工具。QADirector：测试的组织设计和创建以及管理工具。TrackRecord：集成的缺陷跟踪管理工具。EcoTools：高层次的性能监测工具。

优点：

Compuware 的 QACenter 家族集成了一些强大的自动工具，这些工具符合大型计算机应用的测试要求，使开发组获得一致而可靠的应用性能。

QADirector 使用户能够自动地组织测试资料，包括：自动脚本，由 QAHiperstation 或者 QARun 产生手工脚本，测试步骤的序列 MVS 批处理作业脚本，执行作业所必需的 JCL 其他脚本，在用户工作站上执行程序的测试脚本建立测试过程，以便对多种情况和条件进行测试，按正确的次序执行多个测试脚本，记录、跟踪、分析和记录测试结果，与多个并发用户共享测试信息。

是否免费：否

3. 测试管理工具

测试管理工具用于对测试进行管理。一般而言，测试管理工具对测试需求、测试计划、测试用例、测试实施进行管理，并且测试管理工具还包括对缺陷的跟踪管理。测试管理工具能让测试人员、开发人员或其他的 IT 人员通过一个中央数据仓库，在不同地方就能交互信息。测试管理工具的代表有：HP 公司的 TestDirector/Quality Center，Rational 公司的 Test Manager、TestLink，Compureware 公司的 QADirector、SilkCentral Test Manager (SilkPlan Pro)，上海泽众软件公司的 TestCenter 等软件。

1) (HP) TestDirector/Quality Center

功能：业界第一个基于 Web 的测试管理系统，它可以在公司组织内进行全球范围内测试的协调。通过在一个整体的应用系统中提供并且集成了测试需求管理，测试计划，测试日程控制以及测试执行和错误跟踪等功能，TestDirector 极大地加速了测试过程。8.0 后改称 QC。由于 HP QC 可以单独使用 Defect 模块，也可单独申请 Defect 模块的 License，它也是名副其实的缺陷管理工具。

优点：

(1) TD 是一个测试管理工具，管理需求、测试用例设计、测试用例执行、缺陷。而 TM 不涉及需求和缺陷，当然 TM 可以和 RequsitePro，CQ 结合，但毕竟不怎么方便。

(2) TD 可以统计测试用例对需求的覆盖，测试用例对缺陷的覆盖。

(3) 可以和 VSS 集合，对测试用例进行版本控制。

(4) 可以和 CQ 集合，达到两个缺陷工具数据同步。

缺点：

(1) 对 Close 的测试集没有状态标记。

(2) 快速执行测试用例的时候，不能看到测试用例内容。一般快速执行是比较常用的。

(3) 不能实现自动多级联动，需要硬编码。

(4) 没有 System and Browser Matrix。

(5) 报 Defect 的时候，Assign To 只显示开发人员。

是否免费：否

2) (Rational) TestManager

功能：是针对测试活动管理、执行和报告的中央控制台。它是为可扩展性而构建的，支持的范围从纯人工测试方法到各种自动化范型(包括单元测试、功能回归测试和性能测试)。Rational TestManager 可以由项目团队的所有成员访问，确保了测试覆盖信息、缺陷倾势和应用程序准备状态的高度可见性。

优点：

(1) 有自己的客户端，响应速度会快一点儿。

(2) 强大 Case 的管理功能，可以任意条件组合自己的 Test Case。

(3) 可以区分不同的 build 保存测试的结果。

(4) 集成强大的数据生成器(dataPool)。

(5) 与第三方报表完全集成，生成多种且完整的数据报表及图表。

(6) 与 IBM 公司的其他组件(如 CQ,Robot,RequsitePro 等)无缝结合,实现完整的测试管理及测试执行平台。

缺点:

(1) 没有权限管理功能。

(2) 支持的数据库类型太少,尤其 Access 数据库,不能支持太多的访问并发用户数。

(3) 不支持 Test Case 的实例化功能。

(4) 不能预先安排 Case 的执行任务给相应的测试人员。

是否免费:否

3) TestLink

功能:TestLink 是 SourceForge 的开放源代码项目之一。作为基于 Web 的测试管理系统,TestLink 的主要功能包括测试需求管理、测试用例管理、测试用例对测试需求的覆盖管理、测试计划的制订、测试用例的执行、大量测试数据的度量和统计功能等。

优点:

(1) TestLink 用于进行测试过程中的管理,通过使用 TestLink 提供的功能,可以将测试过程从测试需求、测试设计到测试执行完整地管理起来。

(2) 提供了多种测试结果的统计和分析,使用户能够简单地开始测试工作和分析测试结果。

(3) TestLink 可以关联多种 Bug 跟踪系统,如 Bugzilla、Mantis 和 Jira。

缺点:

(1) 不能根据优先级筛选用例,如果需要优先级,必须通过关键字来实现,比较麻烦。

(2) 不能设定测试用例的种类,如果需要必须通过关键字来实现,更麻烦,也不太现实。

(3) 如果测试用例需要大量的数据,创建测试用例时不方便。

使用环境:Apache,MySQL,PHP

是否免费:开源免费

4) (Compureware)QADirector

功能:

(1) 计划和组织测试需求,从多种多样的开发工具和自动测试工具执行测试,在测试过程中允许使用手动测试,观察和分析测试结果,方便地将信息加载到缺陷跟踪系统,针对需求验证应用测试,将分析过程与测试过程结合,确保测试计划符合最终用户需求。

(2) 设计综合测试。QADirector 合理地按可视树状结构组织测试,使测试人员能够建立测试套件,套件中包括大量必需的测试脚本。当测试过程难以自动执行或没有自动化测试工具时,QADirector 提供一个 Internet 浏览器界面,QA 管理人员可以通过这些建立一个综合的手工测试。测试人员可以通过 Internet 浏览器查看哪些手工测试是分给自己的或从未分配的测试中选择相关的测试任务。

(3) 共享和维护集中的测试资产。利用 QADirector,所有测试资产可以通过基于 Aclearcase/" target="_blank" >ccess、SQL Server 或 Oracle 7 和 Oracle 8 的集中存储库进行访问和共享。

(4) 自动地执行测试。QADirector 自动运行多个测试周期,并将结果存储在存储库中。测试可以交互执行或按时间表执行,在预定的日、星期或月完全自动批处理执行。

(5) 测试期间诊断应用问题。在测试周期里诊断应用问题并获得诊断信息是困难的，这对产品在应用中保持性能平稳也是至关重要的。ActiveAnalysis 是 QACenter 的一个新功能，利用它很容易发现和查找测试中的错误。

(6) 快速错误再现。测试和开发环境的不同使再现错误成为一个挑战，例如，在一个复杂的测试场景中的内存泄漏。NuMeGa BoundsChecker Visual C++Edition 是 Compuware 的程序分析工具，用于检测和诊断代码缺陷。BoundsChecker 与 QADirector 的集成使得测试人员能够定位错误。这种错误通常到投产阶段才会被发现。

(7) 故障检查。有故障的程序进入投产阶段结果是难以想象的，需要耗费大量时间来修正。将故障检测能力结合到测试过程，允许测试人员在常规的测试活动期间，自动收集大量关于未覆盖的故障信息。

(8) 性能调试 QADirector 和 NuMega TrueTime 一起工作可以帮助开发人员确定程序为什么运行得慢。TrueTime 是为 Visual C++开发人员准备的一个性能分析工具，它自动定位运行缓慢的代码和性能瓶颈。在 QADirector 报告中，这些数据可在应用投产之前用于调试应用和部件，以使企业的关键应用保持高性能。

(9) 容易地访问和比较测试结果。QADirector 自动存储每次测试运行的结果并从多个工具中整理测试结果。在一个单一的视图中，测试人员可以看到测试脚本的通过/失败状态，快速识别失败并追踪问题。测试结果结合代码覆盖率统计提供了另外一个尺度来评估应用的质量。

(10) 从多个源跟踪缺陷。在测试过程中识别的缺陷可以从 QADirector 加载到 TrackRecord(Compuware 的自动缺陷跟踪工具)。QADirector 把缺陷的 ID 与测试套件一起存储以方便继续跟踪。测试结果也可以从其他源加载，包括 QACenter 和 File-AID 工具。

优点：分布式的测试能力和多平台支持，能够使开发和测试团队跨越多个环境控制测试活动，QADirector 允许开发人员、测试人员和 QA 管理人员共享测试资产，测试过程和测试结果、当前的和历史的信息，从而为客户提供了最完全彻底的、一致的测试。

是否免费：否

5) SilkCentral Test Manager(SilkPlan Pro)

功能：是一个完整的测试管理软件，用于测试的计划、文档和各种测试行为的管理。它提供对人工测试和自动测试的基于过程的分析、设计和管理功能，此外，还提供了基于 Web 的自动测试功能。这使得 SilkPlan Pro 成为 Segue Silk 测试家族中的重要成员和用于监测的解决方案。在软件开发的过程中，SilkPlan Pro 可以使测试过程自动化，节省时间，同时帮助用户回答重要的业务应用面临的关键问题。

优点：在开发过程中充分综合了测试、无人干预地自动执行测试过程、减少维护成本、完全的过程控制、有效的处理变化、用户可以配置和升级、需求跟踪、基于 Web 的报告和缺陷管理工具结合等。

系统需求：

SilkPlan (测试管理)

Pentium 166MHz，64MB RAM，100MB 硬盘空间

Microsoft Windows 95，98，ME，2000，NT4(SP4 或者更高)

Microsoft Access (97,2000),SQL Server(6.5,7.0,2000),Oracle (7.3.4,8.1.6.0).

SilkScheduler Server 组件(自动化测试)

Pentium 500MHz,128MB RAM,150MB 硬盘空间

Microsoft Windows 2000,NT4(SP6 或者更高)

Microsoft SQL Server(7.0,2000), MSDE

是否免费:否

6)(上海泽众软件)TestCenter

功能:上海泽众软件自主研发的一款功能强大的测试管理工具,它可以帮助用户:实现测试用例的过程管理,对测试需求过程、测试用例设计过程、业务组件设计实现过程等整个测试过程进行管理。

是否免费:有免费版本

4. 性能测试工具

专用于性能测试的工具包括:RadView 公司 TestView 系列测试工具;Microsoft 公司的 WebStress 等工具;针对数据库测试的 TestBytes。MercuryInteractive 的 LoadRunner 是一种适用于各种体系架构的自动负载测试工具,它能预测系统行为并优化系统性能。LoadRunner 的测试对象是整个企业的系统,它通过模拟实际用户的操作行为和实行实时性能监测,来帮助用户更快地查找和发现问题。

1)(RadView)TestView

功能:TestView 系列 Web 性能测试工具和 WebLoad Analyzer 性能分析工具旨在测试 Web 应用和 Web 服务的功能、性能、程序漏洞、兼容性、稳定性和抗攻击性;并且能够在测试的同时分析问题原因和定位故障点。从而为测试工作者提供有力的帮助,加速"开发—测试"循环,提高劳动生产率。

TestView 系列 Web 性能测试软件包含三个模块:WebLoad,WebFT 以及 TestView Manager。TestView Manager 用来定制、管理各种测试活动;WebLoad 模拟多个用户行为进行测试,所测试的是系统性能、容量、稳定性和抗攻击性;WebFT 模仿单一用户行为进行测试,所测试的是系统功能、漏洞、兼容性和稳定性。

模块介绍:

(1) WebLoad

WebLoad 专为测试在大量用户访问下的 Web 应用性能而设计。其控制中心运行在 Windows 2000,XP 和 2003 操作系统上,负载发生模块可以运行在 Windows,Solaris 和 Linux 操作系统上。模拟出来的用户流量可支持.NET 和 J2EE 两种环境。

WebLoad 的测试脚本采用 JavaScript 脚本语言实现,支持在 DOM(Document Object Model)的基础之上,将测试单元组织成树状结构,对 Web 应用进行遍历或者选择性测试。WebLoad 还可以录制用户访问 Web 应用的操作过程,自动生成测试脚本,也可以使用脚本编辑器手工编辑或者修改脚本。

WebLoad 的专利技术可以让用户为系统设定最低可接受性能门限值,并让 WebLoad 采用自增用户数的循环测试方式进行测试,这样 WebLoad 就可以自动测得系统的最大用户容量。

WebLoad 不仅能够测试 Web 性能，还能通过直观的图形用户界面直接连接到数据库，测试数据库性能。还可以测试多种 Internet 协议如 FTP，Telnet，SMTP，POP 等的性能。

WebLoad 还可以模拟 DDOS 攻击。它可以模拟诸如 Tfn、Tfn2K、Trinoo、Smurf、Flitz、Carko、Omega3、Plague 和 TCP Flood（SYN、ACK）、UDP Flood、ICMP Flood（Ping、Host-Unreachable）等攻击。通过模拟 DDOS 攻击可以测试 Web 系统在面临 DDoS 攻击的时候的可用性和反应时间的受影响情况。同时，WebLoad 提供有关 DOS 攻击测试的详细报告，帮助用户分析系统漏洞和弱点，为用户加固系统提供依据。

WebLoad 支持与绝大多数的应用服务器和数据库接口，读取它们送出的错误和调试信息。如 IBM 公司的 WebSphere，Sun 公司的 iPlanet，BEA 公司的 WebLogic，Apache，Oracle，SQL Server 等。再结合前端测试结果，WebLoad 能提供全面的 Web 性能分析报告，使用户能够快速定位瓶颈，发现问题。

（2）WebFT

WebFT 帮助用户对 Web 系统进行快速、有效的功能性测试。它是模拟单用户对网站进行功能测试的。

WebFT 支持三个测试级别：全局，页面和对象。用户可以测试系统或者页面的全部功能，也可以深入细致地测试页面上某个对象的功能。例如，HTML 页面的某个属性，某个嵌入的 Java 对象或者 ActiveX 控件。

WebFT 测试脚本与 WebLoad 的完全一样，也是使用 JavaScript 语言写成，也能够自动生成。因此 WebFT 使用的脚本，也可以在 WebLoad 中使用。

（3）TestView Manager

TestView Manager 用来管理和组织各种规模的测试活动，使用它可以定义任意数量和复杂度的脚本。它可以将各个测试脚本组成一个测试项目，用树状结构来组织脚本的执行次序和相互关系，完全模拟用户访问 Web 的行为。TestView 甚至可以同时运行多种测试平台上的多种测试脚本。

TestView Manager 可以为测试制定任意的执行时间表，时间表一旦制定，测试就可以在指定时间里运行，无须人为干预。用户也可以随时去停止、开始或者修改本来按时间表执行的测试。

TestView Manager 提供多个层面的测试结果分析：从高度综合的分析报告到最底层的测试结果数据都可以呈现在用户眼前。同时，TestView 提供各个报告之间的比较功能，为用户后期的测试分析工作提供便利。

是否免费：否

2）（Logic works）TESTBytes

功能：在数据库开发的过程中，为了测试应用程序对数据库的访问，应当在数据库中生成测试用数据，我们可能会发现当数据库中只有少量的数据时程序可能没有问题，但是当真正投入到运用中产生了大量数据就出现问题了。

TESTBytes 是一个用于自动生成测试数据的强大易用的工具，通过简单的点击式操作，就可以确定需要生成的数据类型（包括特殊字符的定制），并通过与数据库的连接来自动生成数百万行的正确的测试数据，可以 TESTBytes 支持的平台有：Windows NT，Windows 95/98，Windows 3. x。

优点：极大地提高数据库开发人员、QA测试人员、数据仓库开发人员、应用开发人员的工作效率。

缺点：

(1) TestBytes能轻松地处理关键数据的关联，但是忽视了其他数据库的约束。

(2) TestBytes产生不相关数据时，该软件允许随机或者连续给出初始关键值。也可以使用SQL的Where语句来限制不相关数据。不过，如果该数据列不是不相关数据关联的一部分，TestBytes仍然不能够给出数据列设置基数。

是否免费：否

3) (HP)LoadRunner

功能：

(1) 轻松创建虚拟用户

使用LoadRunner的Virtual User Generator，用户能很简便地创立起系统负载。该引擎能够生成虚拟用户，以虚拟用户的方式模拟真实用户的业务操作行为。它先记录下业务流程(如下订单或机票预订)，然后将其转化为测试脚本。

(2) 创建真实的负载

Virtual Users建立起后，用户需要设定负载方案，业务流程组合和虚拟用户数量。用LoadRunner的Controller，能很快组织起多用户的测试方案。Controller的Rendezvous功能提供一个互动的环境，在其中既能建立起持续且循环的负载，又能管理和驱动负载测试方案。

(3) 定位性能问题

LoadRunner内含集成的实时监测器，在负载测试过程的任何时候，都可以观察到应用系统的运行性能。这些性能监测器为用户实时显示交易性能数据(如响应时间)和其他系统组件，包括Application Server，Web Server，网络设备和数据库等的实时性能。这样，用户就可以在测试过程中从客户和服务器的双方面评估这些系统组件的运行性能，从而更快地发现问题。

(4) 分析结果以精确定位问题所在

一旦测试完毕，LoadRunner收集汇总所有的测试数据，并提供高级的分析和报告工具，以便迅速查找到性能问题并追溯缘由。使用LoadRunner的Web交易细节监测器，可以了解到将所有的图像、框架和文本下载到每一网页上所需的时间。另外，Web交易细节监测器分解用于客户端、网络和服务器上端到端的反应时间，便于确认问题，定位查找真正出错的组件。

(5) 重复测试保证系统发布的高性能

负载测试是一个重复过程。每次处理完一个出错情况，用户都需要对应用程序在相同的方案下，再进行一次负载测试。以此检验所做的修正是否改善了运行性能。

优点：

(1) LoadRunner是目前软件负载测试的工业标准。

(2) LoadRunner是通过模拟多个用户并发负载，并进行实时监控的方式来进行测试。

(3) 支持多种协议，包括HTTP、WAP、Winsock、Tuxedo、Oracle等。

(4) 与其他负载测试工具的不同在于，LoadRunner的每一个虚拟用户所占用的系统资

源较少,适合用较少的负载测试机器来达到大规模的负载测试所要求的并发压力。LoadRunner 适用于网络应用的负载测试。

缺点:

(1) 对汉语的编码支持问题:UTF-8/GBK 设置导致有时仅用英文作 web_reg_find 的 Check Point。

(2) LoadRunner 8.1 UDP 方式监控 UNIX 资源导致有断续。

(3) 有时应用 VuGen 录制/回放异常退出程序。

(4) 价格昂贵。

(5) 支持 Jboss/Tomcat/MySQL 等的应用性能数据需要自己实现,实际上监控 Linux 也无可用内存、iowait%、网络流量等指标。

系统要求:LoadRunner 分为 Windows 版本和 UNIX 版本。LoadRunner 的 UNIX 版本仅提供 Load Generator 组件的安装(即 LoadRunner 中的负载生成器)。也就是说,这个负载生成器可以在 UNIX 环境下安装和运行,并提供给 Controller 进行远程管理。但是,脚本的录制和场景的设计必须在 Windows 平台完成。

运行 LoadRunner,内存最好在 128MB 以上,这是 LoadRunner 7.8 的最低要求。内存最好在 512MB 以上,安装 LoadRunner 的磁盘空间至少剩余 500MB。操作系统最好为 Windows 2000。

是否免费:否

7.3 测试工具的选择方法

面对目前众多的测试工具,在对其进行分析和评估时,应注重其特性,针对测试的实际需求,可以着重从以下几点入手。

1. 选择标准

选择测试工具并非测试功能越强大越好,因为解决问题是前提,适用才是根本。首先,测试工具要具有跨平台和对环境的兼容性,能够支持不同的运行平台,如各种操作系统和浏览器。其次,操作界面要具有友好型,易使用,功能针对性强;再次,要支持脚本语言,并且具有脚本开发的良好环境,能提供较强的脚本调试功能,可以有效地对测试脚本进行跟踪、错误定位;最后,也要具有报表功能,因为测试结果可能会通过一些统计图来表示,如果有图表表示功能,给人的结果会更直观,更有说服力,容易完成对测试结果的分析和解释。

2. 评估方法

那么,面对目前存在的如此多的测试工具,该怎样分析和评估一个测试工具呢?应该着重从以下几点入手。

功能:选择一个测试工具首先就是看它提供的功能。当然,这并不是说测试工具提供的功能越多就越好,在实际的选择过程中,适用才是根本。

价格:对测试工具的分析评价的主要目的之一,就是选择一个性价比较高的产品。除了功能必须满足需求外,价格是另一个重要的考虑因素。测试工具往往价格昂贵,所以价格

必须在能承受的范围中。另外,借助合适的开源软件测试工具,同样可以构造完整的自动化测试解决方案,又能大大降低成本,也是一种理想的选择。

测试工具的集成能力:测试工具的引入是一个长期的过程,应该是伴随着测试过程改进而进行的一个持续的过程。因此,测试工具的集成能力也是必须考虑的因素,这里的集成包括两个方面的意思,首先,测试工具能否和开发工具进行良好的集成;其次,测试工具能够和其他测试工具进行良好的集成。

测试工具的易用性:利用软件测试工具实现自动化测试要求软件测试人员对测试工具有全面深入的了解,只有很好地熟悉并灵活运用测试工具才能真正发挥测试工具的作用,因而测试工具的易用性对于软件测试人员对工具的掌握程度也是很关键的。

测试工具的侧重点:相当一部分测试工具都有它自己的侧重点,如工具要求的编码规则、采用的代码质量标准、适用的测试对象、适用测试环境等,因而需要考虑测试工具的侧重点是能否满足公司的实际需要。

测试工具引入的连续性和一致性:测试工具是测试自动化的一个重要步骤之一,在引入/选择测试工具时,必须考虑测试工具引入的连续性。也就是说,对测试工具的选择必须有一个全盘的考虑,分阶段、逐步地引入测试工具。

测试工具的技术支持:引进应用软件测试工具是一个长期的过程,当工具引进之后,其开发的公司或单位能否提供良好的技术支持如培训测试人员、提供详尽的使用说明、实例,能否提供持续的升级改进等,是引进工具需要考虑的问题。如人们熟悉的IBMRational公司的Robot+TestManager拥有较多的技术人员,并且还有其他的服务提供商给予技术支持。

3. 选择误区

不同的测试工具具有各自的特点和适用范围,面向的对象也不同,因此并不是任何一个测试工具都能适应每个公司的需求。有些公司花巨资引进了市场上最昂贵的测试工具,但是并不适合本公司的实际情况,最后只能成为摆设而已。因此,在选择测试工具的时候,要本着适用的原则,针对自己的软件产品,能够解决测试过程中最突出的问题。

思考题

1. 测试工具可以分为几类?
2. 请列举常见的白盒测试工具软件名称,并加以对比。
3. 请列举常见的黑盒测试工具软件名称,并加以对比。
4. 请列举常见的测试管理工具软件名称,并加以对比。
5. 请列举常见的性能测试工具软件名称,并加以对比。
6. 测试工具的选择方法有哪几种?

第8章 黑盒测试工具与白盒测试工具

本章学习重点

- 了解使用黑盒测试工具或者白盒测试工具进行软件测试的侧重点。
- 熟悉如何使用 QTP 软件进行黑盒测试。
- 熟悉如何使用 JUnit 软件进行白盒测试。

本章学习难点

掌握如何使用 JUnit 软件进行白盒测试。

8.1 黑盒测试工具

黑盒测试也称功能测试,它是通过测试来检测每个功能是否都能正常使用。在测试中,把程序看作一个不能打开的黑盒子,在完全不考虑程序内部结构和内部特性的情况下,在程序接口进行测试,它只检查程序功能是否按照需求规格说明书的规定正常使用,程序是否能适当地接收输入数据而产生正确的输出信息。黑盒测试着眼于程序外部结构,不考虑内部逻辑结构,主要针对软件界面和软件功能进行测试。本节以 HP QuickTest Professional 为例,讲解在测试过程中如何使用 QTP 进行自动化测试。

8.1.1 QTP 简介

HP QuickTest Professional 提供符合所有主要应用软件环境的功能测试和回归测试的自动化,采用关键字驱动的理念以简化测试用例的创建和维护。它让用户可以直接录制屏幕上的操作流程,自动生成功能测试或者回归测试用例。专业的测试者也可以通过提供的内置脚本和调试环境来取得对测试和对象属性的完全控制。

QTP 进行功能测试的测试流程大致有以下 5 个步骤。

1. 制定测试计划

自动测试的测试计划是根据被测项目的具体需求,以及所使用的测试工具而制定的,完

全用于指导测试全工程。QTP是一个功能测试工具，主要帮助测试人员完成软件的功能测试。与其他测试工具一样，QTP不能完全取代测试人员的手工操作，但是在某个功能点上，使用QTP的确能够帮助测试人员做很多工作。在测试计划阶段，首先要做的就是分析被测应用的特点，决定应该对哪些功能点进行测试，可以考虑细化到具体页面或者具体控件。对于一个普通的应用程序来说，QTP应用在某些界面变化不大的回归测试中是非常有效的。

2. 创建测试脚本

当测试人员浏览站点或在应用程序上操作的时候，QTP的自动录制机制能够将测试人员的每一个操作步骤及被操作的对象记录下来，自动生成测试脚本语句。与其他自动测试工具录制脚本有所不同的是，QTP除了以VBScript脚本语言的方式生成脚本语句以外，还将被操作的对象及相应的动作按照层次和顺序保存在一个基于表格的关键字视图中。例如，当测试人员单击一个链接，然后选择一个复选框或者提交一个表单，这样的操作流程都会被记录在关键字视图中。

3. 增强测试脚本的功能

录制脚本只是实现创建或者设计脚本的第一步，基本的脚本录制完毕后，测试人员可以根据需要增加一些扩展功能，QTP允许测试人员通过在脚本中增加或更改测试步骤来修正或自定义测试流程，如增加多种类型的检查点功能，既可以让QTP检查在程序的某个特定位置或对话框中是否出现了需要的文字，还可以检查一个链接是否返回了正确的URL地址等。还可以通过参数化功能，使用多组不同的数据驱动整个测试过程。

4. 运行测试

QTP从脚本的第一行开始执行语句，运行过程中会对设置的检查点进行验证，用实际数据代替参数值，并给出相应的输出结构信息。测试过程中，测试人员还可以调试自己的脚本，直到脚本完全符合要求。

5. 分析测试结果

运行结束后系统会自动生成一份详细完整的测试结果报告。

Mercury QuickTest Professional甚至可以使新测试人员在几分钟内提高效率。测试人员只需通过单击“记录”按钮，并使用执行典型业务流程的应用程序即可创建测试脚本。系统使用简明的英文语句和屏幕抓图来自动记录业务流程中的每个步骤。用户可以在关键字视图中轻松修改、删除或重新安排测试步骤。QuickTest Professional可以自动引入检查点，以验证应用程序的属性和功能，例如，验证输出或检查链接有效性。对于关键字视图中的每个步骤，活动屏幕均准确显示测试中应用程序处理此步骤的方式。也可以为任何对象添加几种类型的检查点，以便验证组件是否按预期运行（只需在活动屏幕中单击此对象即可）。然后，可以在产品介绍（具有Excel所有功能的集成电子表格）中输入测试数据，以便在不需要编程的情况下处理数据集和创建多个测试迭代，从而扩大测试案例范围。可以输入数据，或从数据库、电子表格或文本文件中导入数据。

高级测试人员可以在专家视图中查看和编辑自己的测试脚本，该视图显示 QuickTest Professional 自动生成的基于业界标准的内在 VB 脚本。专家视图中进行的任何变动自动与关键字视图同步。

一旦测试人员运行了脚本，TestFusion 报告显示测试运行的所有方面：高级结果概述，准确指出应用程序故障位置的可扩展树视图，使用的测试数据，突出显示任何差异的应用程序屏幕抓图，以及每个通过和未通过检查点的详细说明。通过使用 Mercury TestDirector 合并 TestFusion 报告，可以在整个 QA 和开发团队中共享报告。

QuickTest Professional 也加快了更新流程。当测试中应用程序出现变动(例如“登录”按钮重命名为“登入”)时，可以对共享对象库进行一次更新，然后此更新将传播到所有引用该对象的脚本。可以将测试脚本发布到 Mercury TestDirector，使其他 QA 团队成员可以重复使用测试脚本，从而消除了重复工作。

8.1.2 录制测试脚本

1. 录制测试脚本

Mercury Tours 示范网站是一个提供机票预订服务的网站，接下来，使用 MI 公司提供的 Mercury Tours 示范网站作为演示 QuickTest 各个功能的例子程序。

(1) 执行 QuickTest 并开启一个全新的测试脚本。

开启 QuickTest，在 Add-in Manager 窗口中选择 Web 选项，单击 OK 按钮关闭 Add-in Manager 窗口，进入 QuickTest Professional 主窗口。

如果 QuickTest Professional 已经启动，检查 Help→About QuickTest Professional 查看目前加载了哪些 add-ins。如果没有加载 Web，那么必须关闭并重新启动 QuickTest Professional，然后在 Add-in Manager 窗口中选择 Web。

如果在执行 QuickTest Professional 时没有开启 Add-in Manager，则单击 Tool→Options，在 General 选项卡中勾选 Display Add-in Manager on Startup，在下次执行 QuickTest Professional 时就会看到 Add-in Manager 窗口。

(2) 开始录制测试脚本。

选中 Test→Record 或者单击工具栏上的 Record 按钮，打开 Record and Run Settings 对话框，如图 8-1 所示。

在 Web 选项卡中选择 Open the following browser when a record or run session begins 单选按钮，在 Type 下拉列表中选择 Microsoft Internet Explorer 为浏览器的类型；在 Address 中添加 http://newtours. mercuryinteractive. com/。这样，在录制的时候，QuickTest 会自动打开 IE 浏览器并连接到 Mercury Tours 范例网站上。

现在切换到 Windows Applications 选项卡，如图 8-2 所示。

如果选择 Record and run test on any open Windows-based application 单选按钮，则在录制过程中，QuickTest 会记录用户对所有的 Windows 程序所做的操作。如果选择 Record and run on these applications (opened a session start) 单选按钮，则在录制过程中，QuickTest 只会记录对那些添加到下面 Application details 列表框中的应用程序的操作(可以通过 Add、Edit、Delete 按钮来编辑这个列表)。

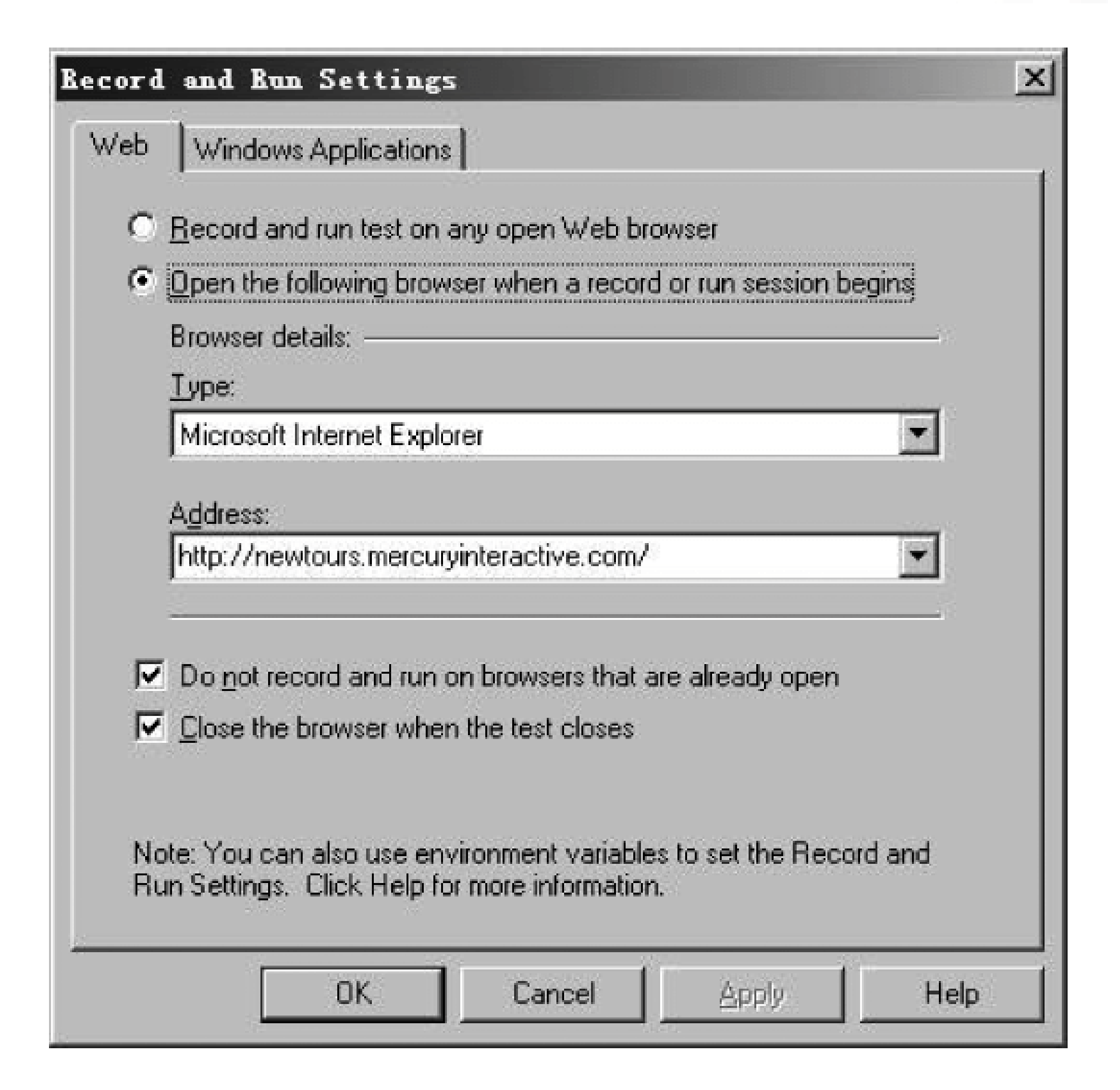


图 8-1　Record and Run Settings 对话框

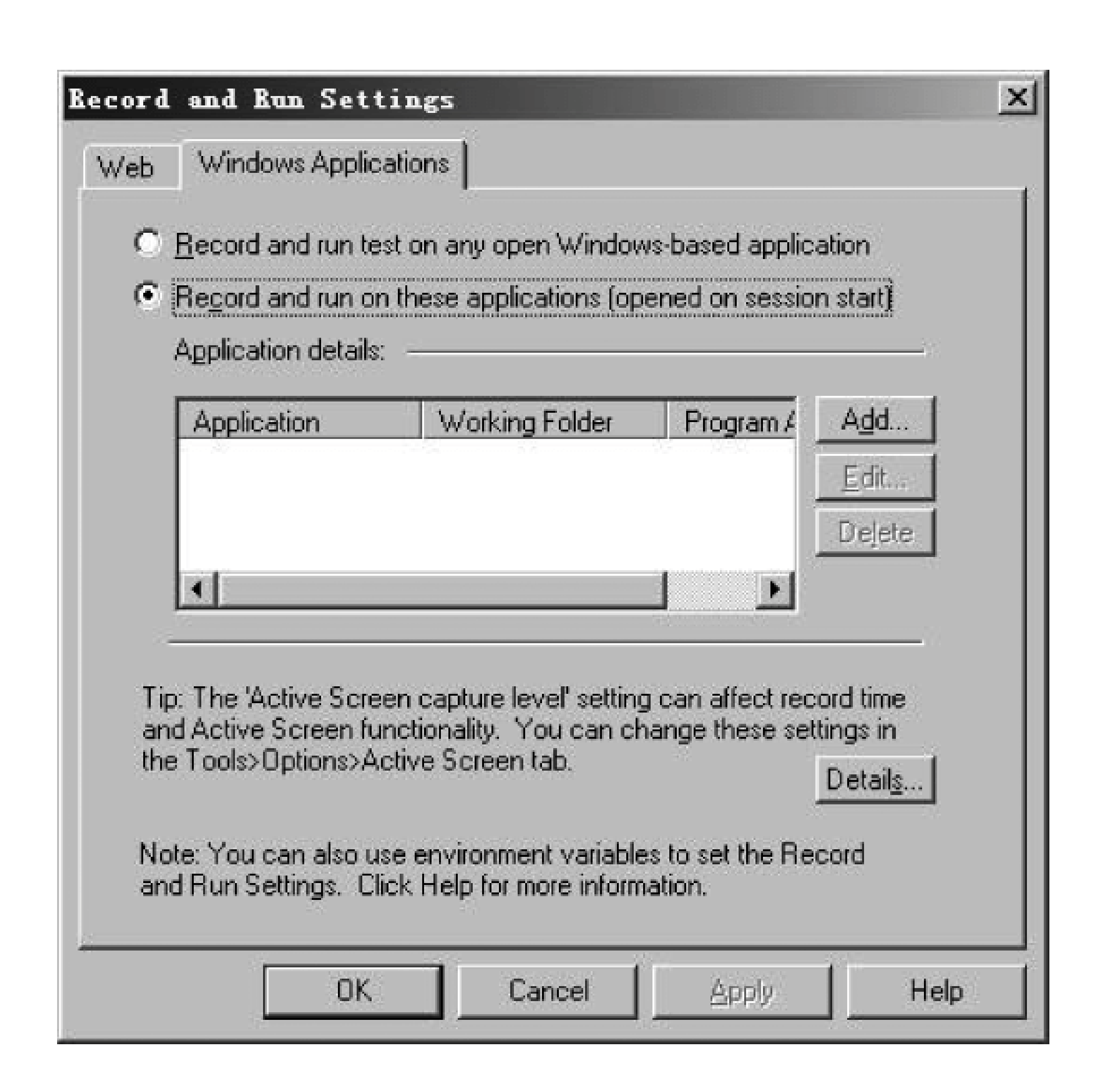


图 8-2　Windows Applications 选项卡

这里选择第二个单选按钮。因为这里只是对 Mercury Tours 范例网站进行操作，不涉及 Windows 程序，所以保持列表为空。

单击 Apply 按钮，开始录制，将自动打开 IE 浏览器并连接到 Mercury Tours 范例网站上。

(3) 登录 Mercury Tours 网站。

在用户名和密码文本框中输入注册时使用的账号和密码，单击 Sign-in，进入 Flight Finder 网页。

(4) 输入订票数据。

输入以下订票数据：

Departing From：New York

On：May 14

Arriving In：San Francisco

Returning：May 28

Service Class：Business class

其他字段保留默认值，单击 CONTINUE 按钮打开 Select Flight 页面。

(5) 选择飞机航班。

可以保存默认值，单击 CONTINUE 按钮打开 Book a Flight 页面。

(6) 输入必填字段(红色字段)。

输入用户名和信用卡号码(信用卡可以输入虚构的号码，如 8888-8888)。

单击网页下方的 SECURE PURCHASE 按钮，打开 Flight Confirmation 网页。

(7) 完成定制流程。

查看订票数据，并选择 BACK TO HOME 回到 Mercury Tours 网站首页。

(8) 停止录制。

在 QuickTest 工具列上单击 Stop 按钮，停止录制。

到这里已经完成了预订从纽约到旧金山机票的动作，并且 QuickTest 已经录制了从单击 Record 按钮后到 Stop 按钮之间的所有操作。

(9) 保存脚本。

选择 File→Save 或者单击工具栏上的 Save 按钮，开启 Save 对话框。选择路径，填写文件名，这里取名为 Flight。单击"保存"按钮进行保存。

通过以上 9 个步骤，录制了一个完整的测试脚本——预订从纽约到旧金山的机票。

2. 分析录制的测试脚本

在录制过程中，QuickTest 会在测试脚本管理窗口(也叫 Tree View 窗口)中产生对每一个操作的相应记录，并在 Keyword View 中以类似 Excel 工作表的方式显示所录制的测试脚本。当录制结束后，QuickTest 也就记录下测试过程中的所有操作。测试脚本管理窗口显示的内容如图 8-3 所示。

Item	Operation	Value	Assignment	Comment	Documentation
Action1					
Welcome: Mercury Tours					
Welcome: Mercury Tours					
userName	Set	"lgh1"			Enter "lgh1" in the "userName" edit box.
password	SetSecure	"446845bf84444adc2...			Enter the encrypted string "446845bf84444adc245d5e096636301a520d" in the "password" e...
Sign-In	Click	41,4			Click the "Sign-In" image.
Find a Flight: Mercury					
fromPort	Select	"New York"			Select the "New York" item in the "fromPort" list.
toPort	Select	"San Francisco"			Select the "San Francisco" item in the "toPort" list.
toDay	Select	"28"			Select the "28" item in the "toDay" list.
servClass	Select	"Business"			Select radio button "Business" in the "servClass" radio button group.
findFlights	Click	70,12			Click the "findFlights" image.
Select a Flight: Mercury					
reserveFlights	Click	51,12			Click the "reserveFlights" image.
Book a Flight: Mercury					
passFirst0	Set	"l"			Enter "l" in the "passFirst0" edit box.
passLast0	Set	"gh"			Enter "gh" in the "passLast0" edit box.
creditnumber	Set	"8888-8888"			Enter "8888-8888" in the "creditnumber" edit box.
buyFlights	Click	82,17			Click the "buyFlights" image.
Flight Confirmation: Mercury					
home	Click				Click the "home" image.
Welcome: Mercury Tours_2	Sync				Wait for the Web page to synchronize before continuing the run.

Keyword View / Expert View

图 8-3 测试脚本管理窗口

在 Keyword View 中的每一个字段都有其意义。

(1) Item：以阶层式的图标表示这个操作步骤所作用的组件(测试对象、工具对象、函数呼叫或脚本)。

(2) Operation：要在这个作用到的组件上执行的动作，如单击、选择等。

(3) Value：执行动作的参数，例如，当鼠标单击一张图片时是用左键还是右键。

(4) Assignment：使用到的变量。

(5) Comment：在测试脚本中加入的批注。

(6) Documentation：自动产生用来描述此操作步骤的英文说明。

脚本中的每一个步骤在 Keyword View 中都会以一列来显示，其中包含用来表示此组件类别的图标以及此步骤的详细数据。

下面针对一些常见的操作步骤做详细说明，如表 8-1 所示。

表 8-1 常见的操作步骤与说明

步　骤	说　明
Action1	Action1 是一个动作的名称
Welcome: Mercury Tours	Welcome：Mercury 是被浏览器开启的网站的名称
Welcome: Mercury Tours	Welcome：Mercury Tours 是网页的名称
userName　Set　"lgh1"	userName 是 edit box 的名称 Set 是在这个 edit box 上执行的动作 lgh1 是被输入的值
password　SetSecure　"446845bf84444adc2...	password 是 edit box 的名称 SetSecure 是在这个 edit box 上执行的动作，此动作有加密的功能 446845bf84444adc…是被加密过的密码
Sign-In　Click　41,4	Sign-In 是图像对象的名称 Click 是在这个图像上执行的动作 41,4 则是这个图像被单击的 X,Y 坐标

3. 执行测试脚本

当运行录制好的测试脚本时，QuickTest 会打开被测试程序，执行在测试中录制的每一个操作。测试运行结束后，QuickTest 显示本次运行的结果。接下来，执行录制的 Flight 测试脚本。

(1) 打开录制的 Flight 测试脚本。

(2) 设置运行选项。单击 Tool→Options 打开 Options 对话框，选择 Run 选项卡，如图 8-4 所示。

如果要将所有画面储存在测试结果中，在 Save step screen capture to results 选项中选择 Always 选项。一般情况下选择 On errors 或 On error and warning 表示在回放测试过程中出现问题时，才保存图像信息。在这里为了更多地展示 QuickTest 的功能，所以选择使用 Always 选项。

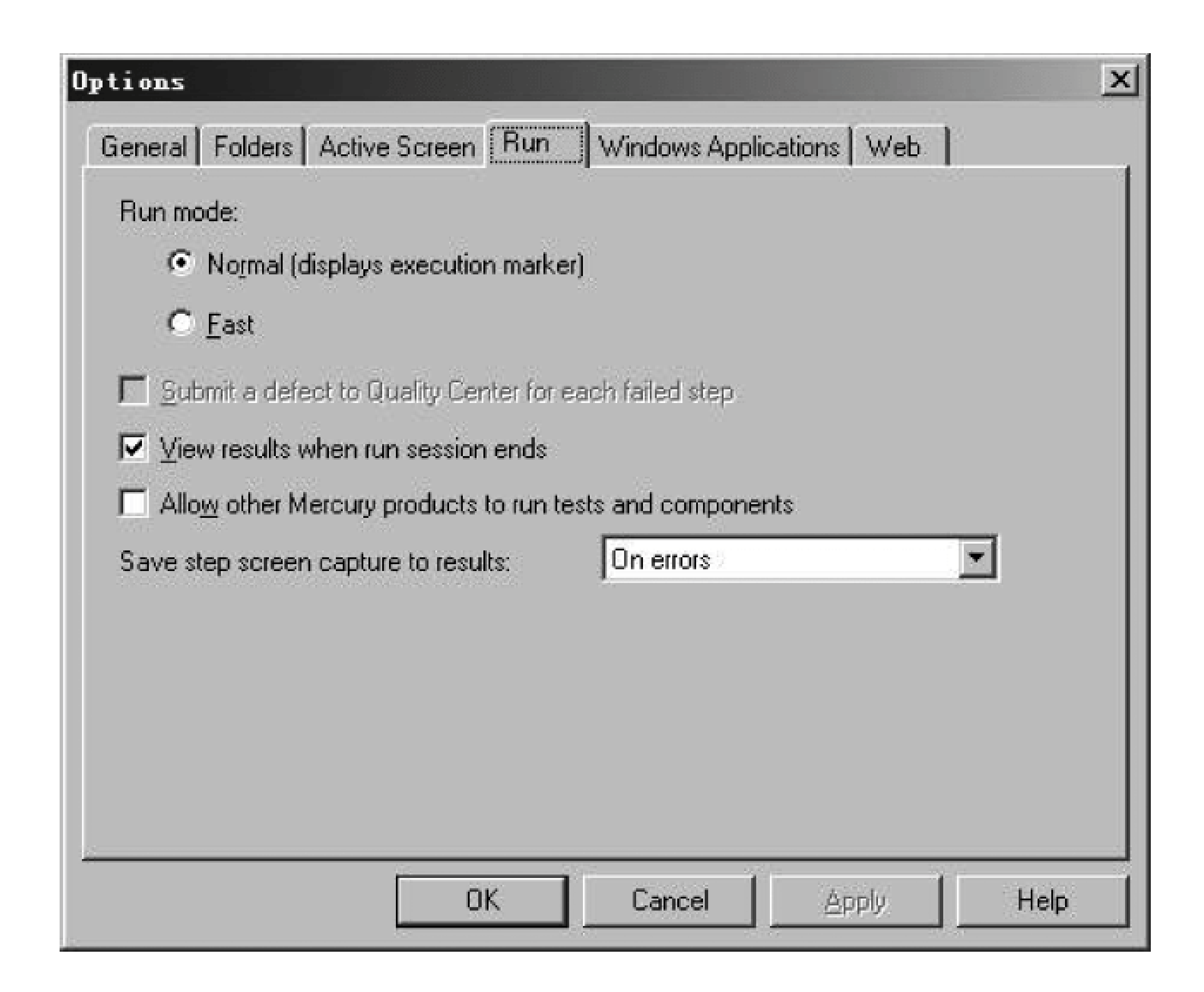


图 8-4 Options 对话框

(3) 在工具条上单击 Run 按钮,打开 Run 对话框,如图 8-5 所示。

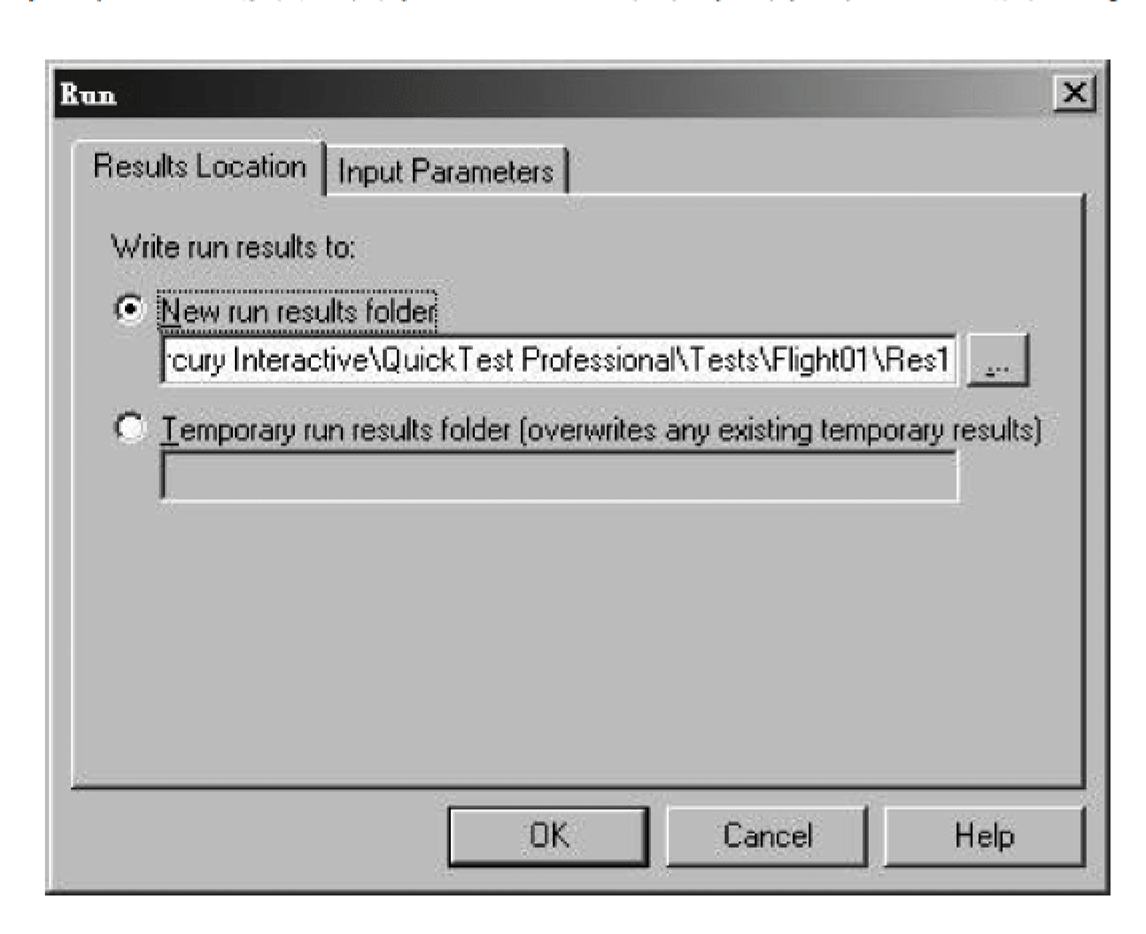


图 8-5 Run 对话框

询问要将本次的测试运行结果保存到何处。选择 New run results folder 单选按钮,设定好存放路径(在这里使用预设的测试结果名称)。

(4) 单击 OK 按钮开始执行测试。

可以看到,QuickTest 按照在脚本中录制的操作,一步一步地运行测试,操作过程与手工操作时完全一样。同时,可以在 QuickTest 的 Keyword View 中出现一个黄色的箭头,指示目前正在执行的测试步骤。

4. 分析测试结果

在测试执行完成后,QuickTest 会自动显示测试结果窗口,如图 8-6 所示。

在这个测试结果窗口中分为两个部分显示测试执行的结果。

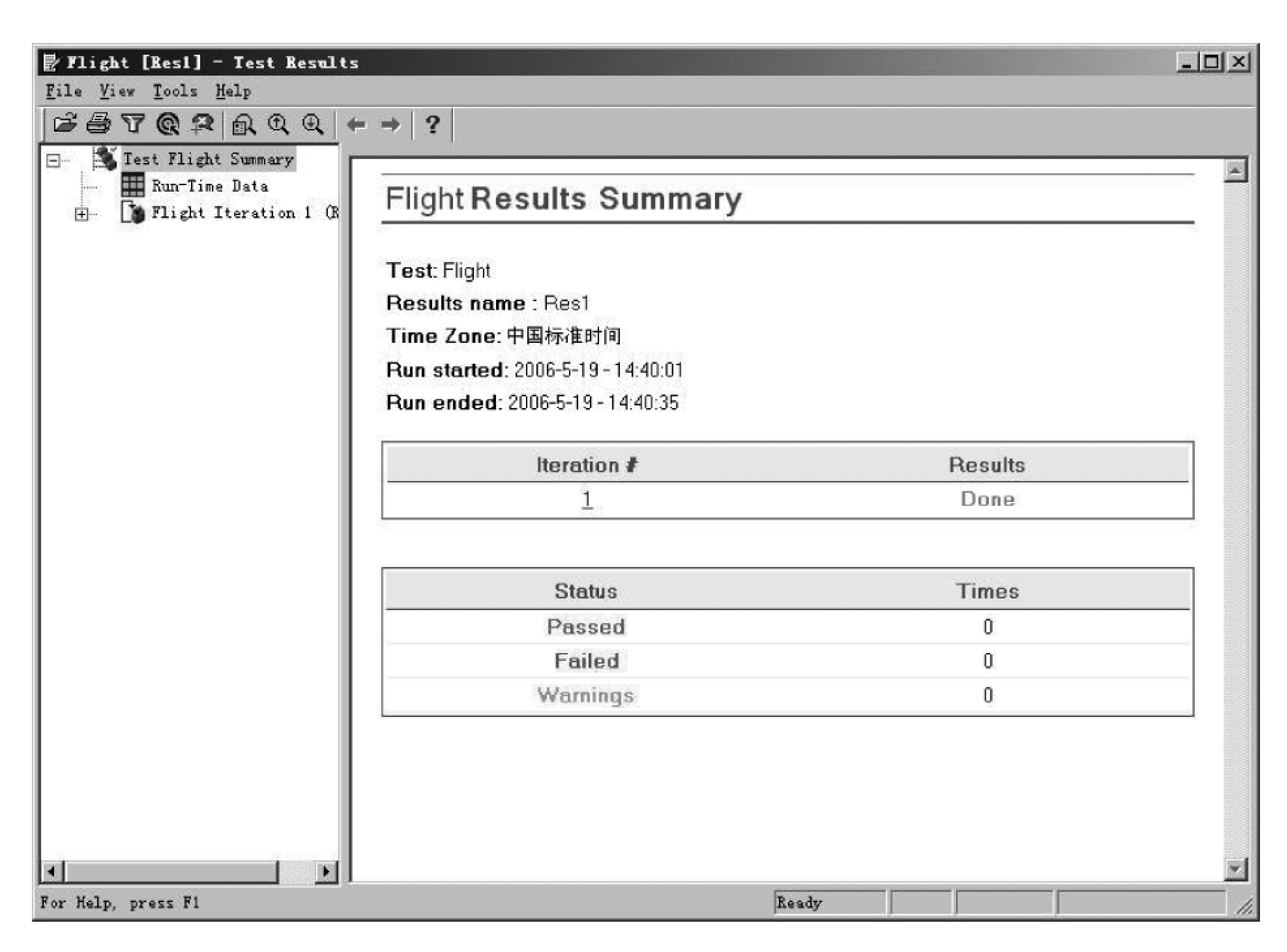


图 8-6　测试结果窗口

(1) 左边以阶层图标的方式显示测试脚本所执行的步骤。可以单击“＋”检查每一个步骤，所有的执行步骤都会以图示的方式显示。可以设定 QuickTest 以不同的方式执行每个测试或某个动作，每执行一次反复称为一个迭代，每一次迭代都会被编号(在上面的例子中只执行了一次迭代)。

(2) 右边则是显示测试结果的详细信息。在第一个表格中显示哪些迭代是已经通过的，哪些是失败的。第二个表格是显示测试脚本的检查点，哪些是通过的，哪些是失败的，以及有几个警告信息。

在上面的测试中，所有的测试都是通过的，在脚本中也没有添加检查点(有关检查点的内容将在以后的课程中学习)。接下来查看 QuickTest 执行测试脚本的详细结果，以及选择某个测试步骤时出现的详细信息。

在树视图中展开 Flight Iteration 1(Row 1)→Action1 Summary→Welcome Mercury Tours→Find a Flight：Mercury，选择"fromPost"：Select "New York"，如图 8-7 所示。

在这个测试结果窗口中显示三个部分，分别如下。

(1) 左边是 Test results tree：展开树视图后，显示了测试执行过程中的每一个操作步骤。选择某一个测试步骤，会在右边区域显示相应的信息。

(2) 右上方是 Test results detail：对应当前选中的测试步骤，显示被选取测试步骤执行时的详细信息。

(3) 右下方是 Active Screen：对应当前选中的测试步骤，显示该操作执行时应用程序的屏幕截图。

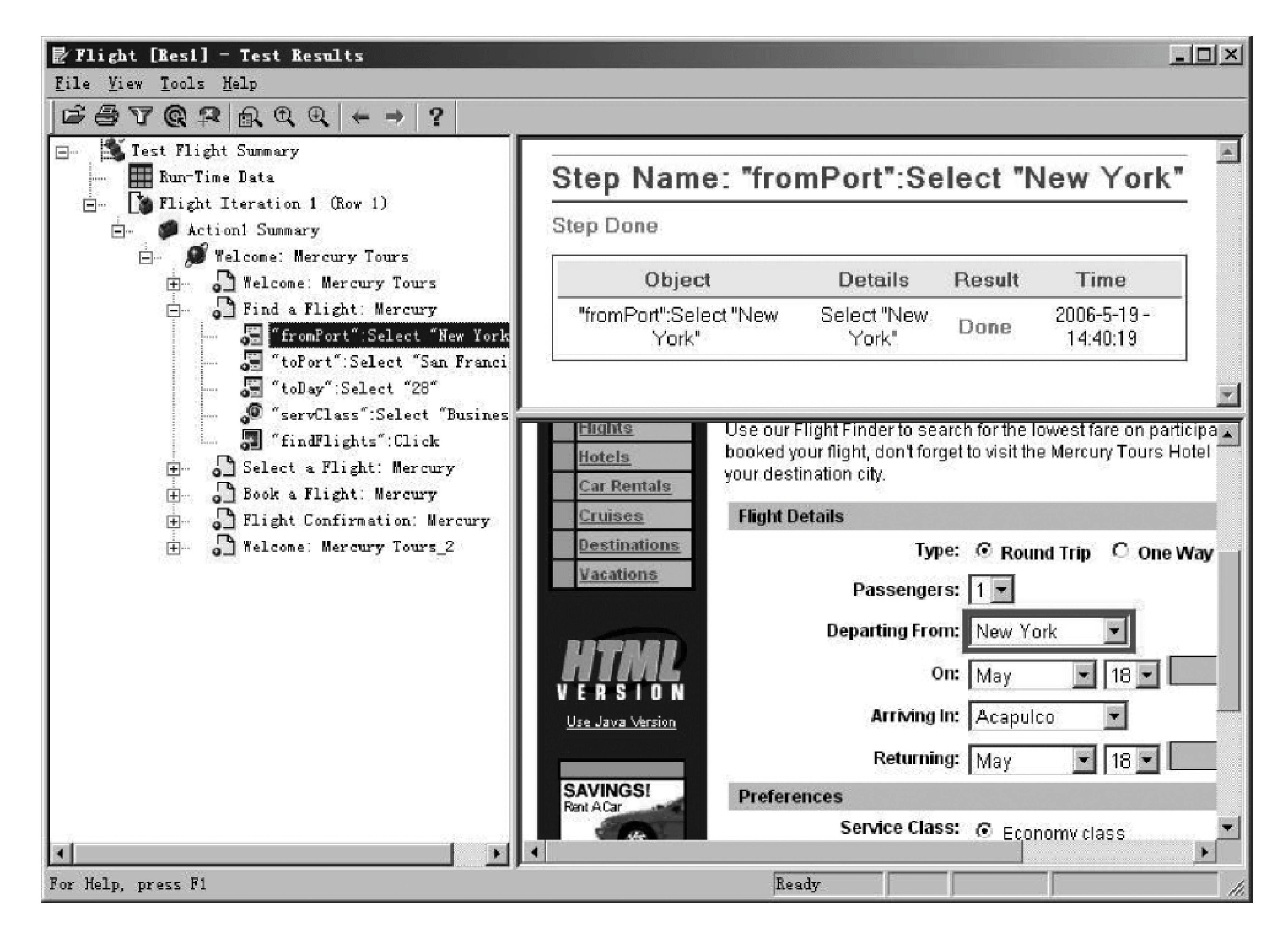


图 8-7 展开树视图

当选中 Test results tree 上的网页图示时，会在 Active Screen 中看到执行时的画面。当选中 Test results tree 上的测试步骤(在某个对象上执行某个动作)时，除了显示当前的画面外，对象还会被粉色的方框框住。在上面的例子中，在 Active Screen 中单击被框住的 Departing From 下拉菜单，会显示其他的选项。

8.1.3 建立检查点

通过 8.1.2 节的学习，读者已经掌握了如何录制、执行测试脚本以及查看测试结果。但是这只是实现了测试执行的自动化，没有实现测试验证的自动化，所以这并不是真正的自动化测试。在这一节将学习如何在测试脚本中设置检查点，以验证执行结果的正确性。

"检查点"是将指定属性的当前值与该属性的期望值进行比较的验证点。这能够确定网站或应用程序是否正常运行。当添加检查点时，QuickTest 会将检查点添加到关键字视图中的当前行并在专家视图中添加一条"检查检查点"语句。运行测试或组件时，QuickTest 会将检查点的期望结果与当前结果进行比较。如果结果不匹配，检查点就会失败。可以在"测试结果"窗口中查看检查点的结果。

1. QuickTest 检查点种类

首先了解一下 QuickTest 支持的检查点种类，如表 8-2 所示。

表 8-2　QuickTest 检查点种类

检查点类型	说　　明	范　　例
标准检查点	检查对象的属性	检查某个按钮是否被选中
图片检查点	检查图片的属性	检查图片的来源文件是否是正确的
表格检查点	检查表格的内容	检查表格内的内容是否是正确的
网页检查点	检查网页的属性	检查网页加载的时间或网页是否含有不正确的链接
文字/文字区域检查点	检查网页上或是窗口上出现的文字是否正确	检查登录系统后是否出现"登录成功"的文字
图像检查点	提取网页和窗口的画面检查画面是否正确	检查网页或者网页的一部分是否如期显示
数据库检查点	检查数据库的内容是否正确	检查数据库查询的值是否正确
XML 检查点	检查 XML 文件的内容	XML 检测点有两种——XML 文件检测点和 XML 应用检测点。XML 文件检测点用于检查一个 XML 文件；XML 应用检测点用于检查一个 Web 页面的 XML 文档

可以在录制测试的过程中或录制结束后，向测试脚本中添加检测点。下面学习如何在测试脚本上建立检查点。

2. 创建检查点

打开 Flight 测试脚本，将脚本另存为 Checkpoint 测试脚本。在 Checkpoint 测试脚本中创建 4 个检查点，分别是对象检查、网页检查、文字检查以及表格检查。

1）对象检查

通过向测试或组件中添加标准检查点，可以对不同版本的应用程序或网站中的对象属性值进行比较。可以使用标准检查点来检查网站或应用程序中的对象属性值。标准检查点将对录制期间捕获的对象属性的预期值，与运行会话期间对象的当前值进行比较。

首先在 Checkpoint 测试脚本上添加一个标准检查点，这个检查点用于检查旅客的姓氏。

创建标准检查点步骤如下。

（1）打开 Checkpoint 测试脚本。

（2）选择要建立检查点的网页。

在 QuickTest 的视图树中展开 Action1→Welcome：Mercury Tours→Book a Flight：Mercury，由于输入使用者姓氏的测试步骤是 passFirst0 这个步骤，所以要选择这个步骤的下一个测试步骤，以便建立检查点，如图 8-8 所示。

（3）建立标准检查点。

对 Active Screen 中的 First Name 编辑框单击鼠标右键，显示插入选择点的类型，如图 8-9 所示。

选择 Insert Standard Checkpoint 选项，显示 Object Selection-Checkpoint Properties 对话框，如图 8-10 所示。

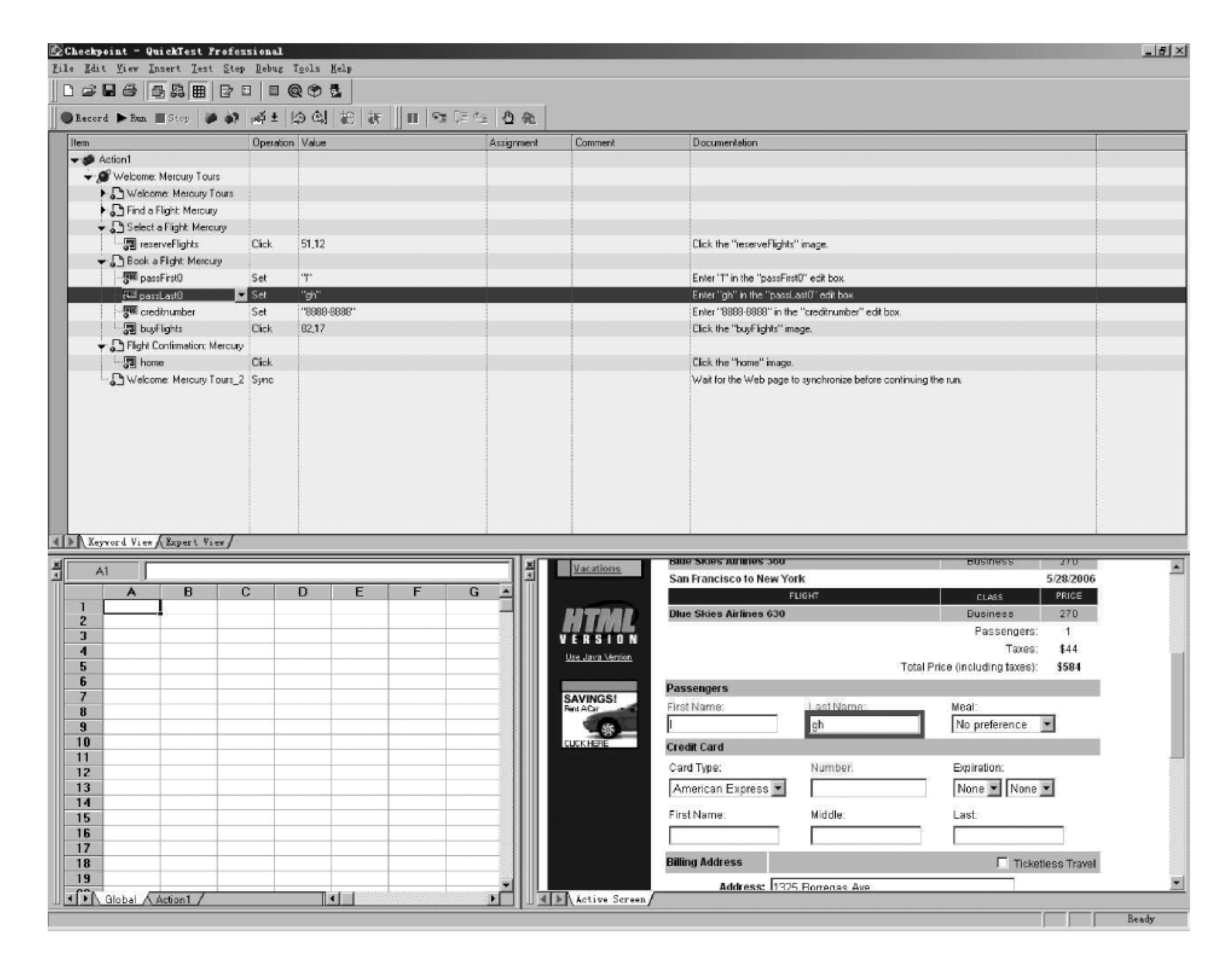


图 8-8　选择要建立检查点的网页

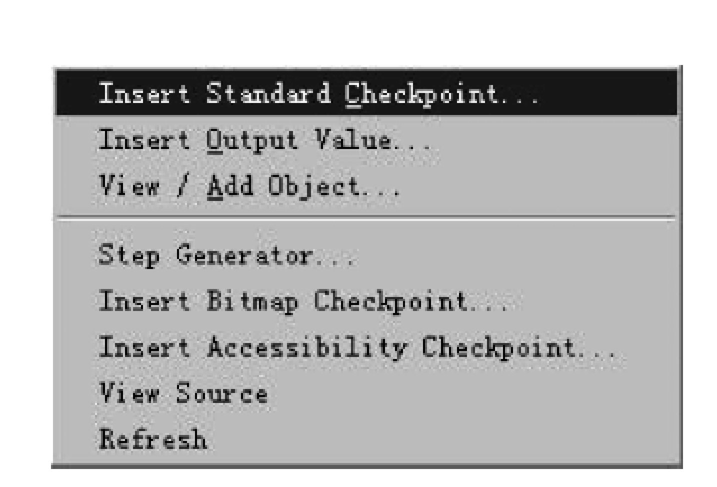


图 8-9　显示插入选择点的类型

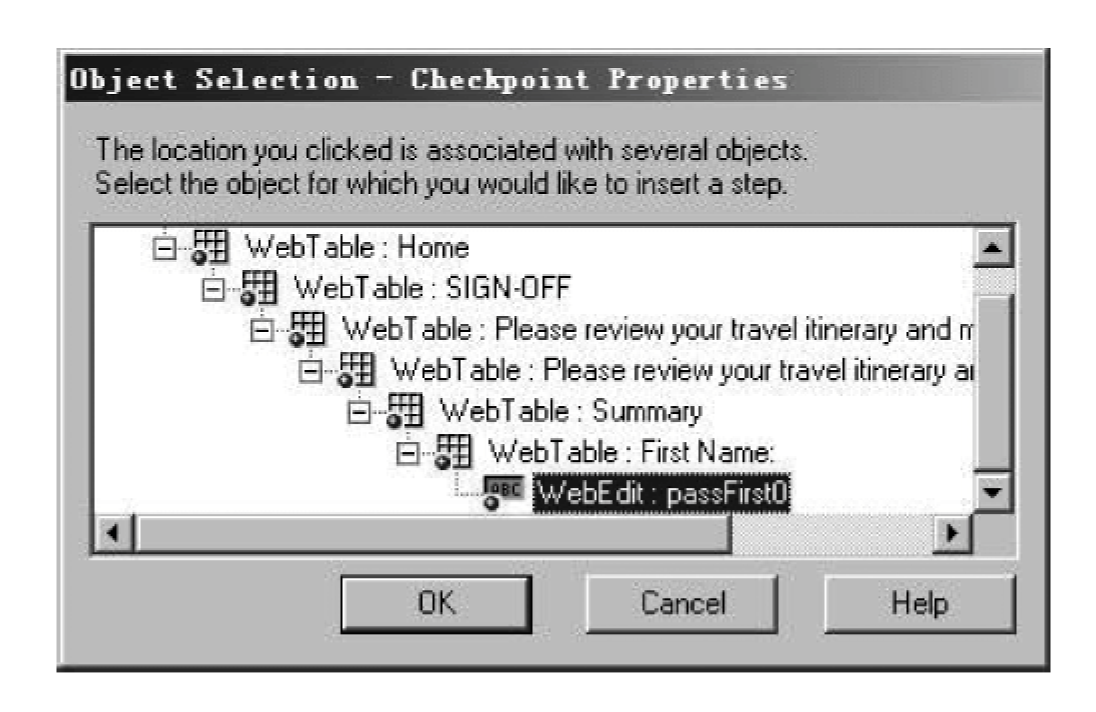


图 8-10　Object Selection-Checkpoint Properties 对话框

确保当前的焦点定位在 WebEdit：passFirst0 上，单击 OK 按钮，弹出如图 8-11 所示的对话框。

在检查点属性对话框中会显示检查点的属性。

① Name：检查点的名称。

② Class：检查点的类别，WebEdit 表示这个检查点是一个输入框。

③ Type 字段中的 ABC 图标：表示这个属性的值是一个常数。

对于每一个检查点，QuickTest 会使用预设的属性作为检查点的属性，表 8-3 说明了这些预设的属性。

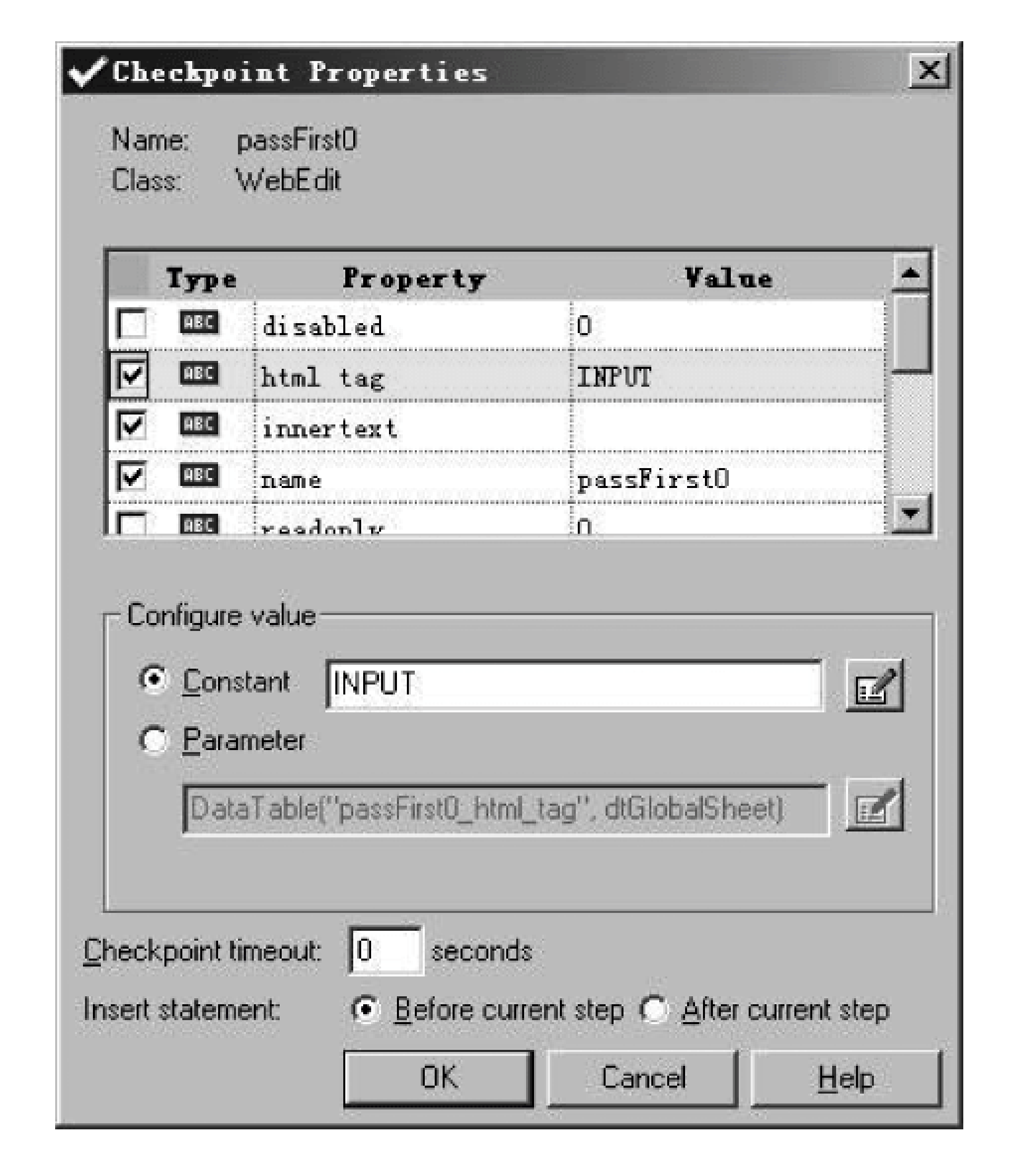


图 8-11　检查点属性

表 8-3　预设属性的说明

属性	值	说　明
html tag	INPUT	HTML 原始码中的 INPUT 标签
innertext		在这个范例中，innertext 只是空的，检查点会检查当执行时这个属性是不是空的
name	passFirst0	passFirst0 是这个编辑框的名称
type	text	text 是 HTML 原始码中 INPUT 对象的类型
value	姓氏（录制脚本时输入的姓氏）	在编辑框中输入的文字

接受预设的设定值，单击 OK 按钮。QuickTest 会在选取步骤之前建立一个标准检查点，如图 8-12 所示。

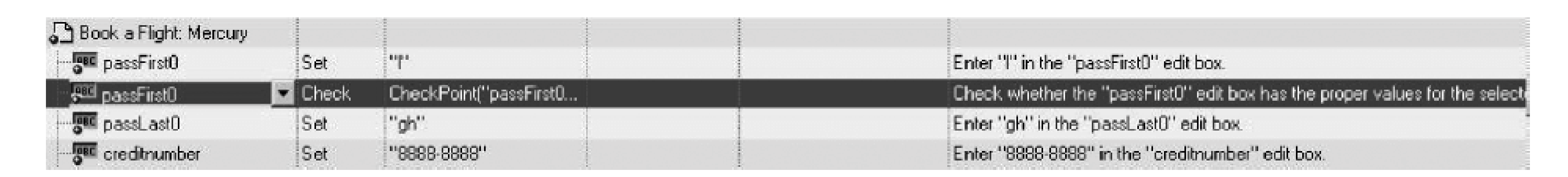


图 8-12　建立一个标准检查点

（4）在工具栏上单击 Save 按钮保存脚本。

通过（1）～（4）的步骤，添加一个标准检查点的操作到此结束。

2）网页检查

在 Checkpoint 测试脚本中再添加一个网页检查点，网页检查点会检查网页的链接以及图像的数量是否与当前录制时的数量一致。网页检查点只能应用于 Web 页面中。

创建网页检查点的步骤如下。

（1）选择要建立检查点的网页。

展开 Action1→Welcome：Mercury Tours，选择 Book a Flight：Mercury 页面，在

Active Screen 中会显示相应的页面。

(2) 建立网页检查点。

在 Active Screen 上的任意地方单击鼠标右键,选中 Insert Standard Checkpoint 选项,开启 Object Selection-Checkpoint Properties 对话框(由于选择的位置不同,对话框中显示被选取的对象可能不一样),如图 8-13 所示。

选择最上面的 Page: Book a Flight: Mercury,并单击 OK 按钮确认,将打开 Page Checkpoint Properties 对话框,如图 8-14 所示。

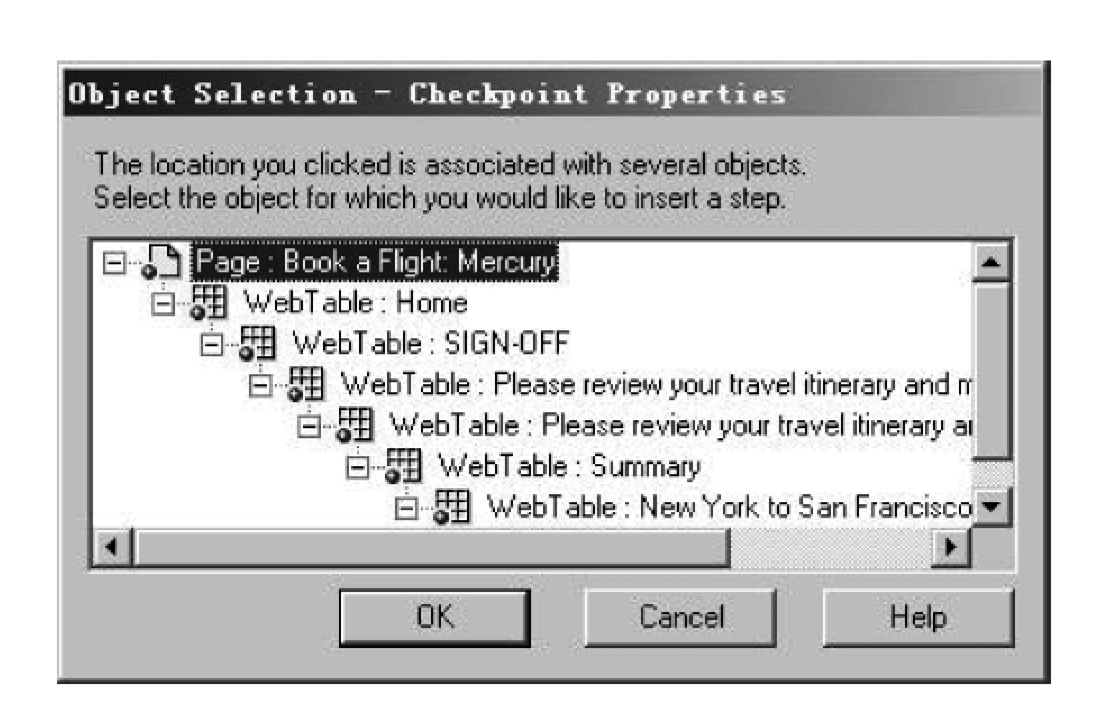


图 8-13 Object Selection-Checkpoint Properties 对话框

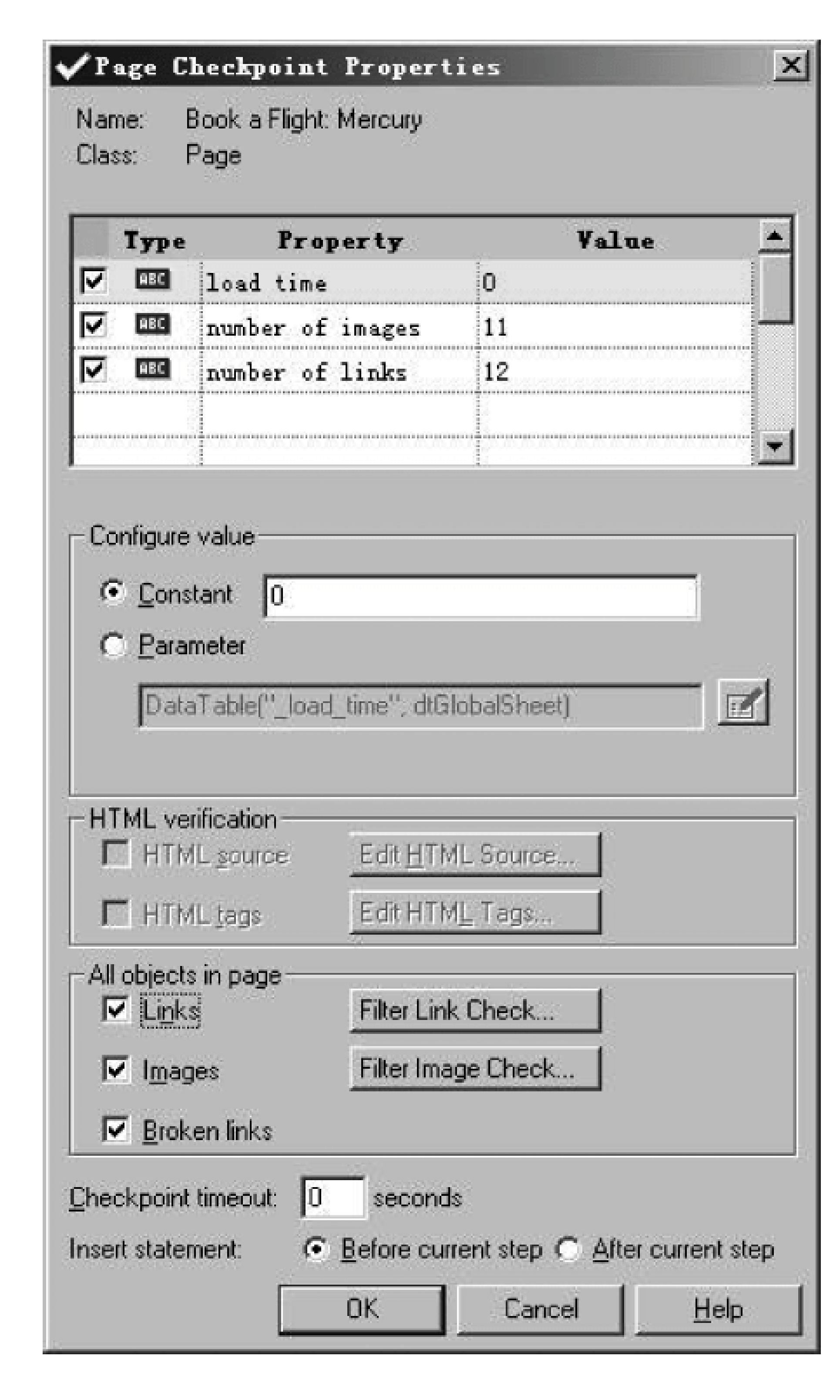


图 8-14 Page Checkpoint Properties 对话框

当执行测试时,QuickTest 会检查网页的链接与图片的数量,以及加载的时间,如同对话框上方所显示的那样。

QuickTest 也检查每一个链接的 URL 以及每一个图片的原始文件是否存在。

(3) 接受默认设定,单击 OK 按钮。QuickTest 会在 Book a Flight: Mercury 网页上加一个网页检查。

(4) 在工具栏上单击 Save 按钮保存脚本。

3) 文字检查

下面建立一个文字检查点,检查在 Flight Confirmation 网页中是否出现“New York”文字。

建立文字检查点的步骤如下。

(1) 确定要建立检查点的网页。

展开 Action1→Welcome: Mercury Tours 选择 Flight Confirmation: Mercury 页面,在 Active Screen 中会显示相应的页面。

(2) 建立文字检查点。

在 Active Screen 中选择在 Departing 下方的 New York。

对选取的文字单击鼠标右键,并选取 Insert Text Checkpoint 选项打开 Text Checkpoint Properties 对话框,如图 8-15 所示。

当 Checked Text 出现在下拉列表中时,在 Constant 字段中显示的就是选取的文字。这也就是 QuickTest 在执行测试脚本时所要检查的文字。

(3) 单击 OK 按钮关闭对话框。

QuickTest 会在测试脚本上加上一个文字检查点,这个文字检查点会出现在 Flight Confirmation: Mercury 网页下方。

(4) 在工具栏上单击 Save 按钮保存脚本。

4) 表格检查

通过添加表检查点,可以检查应用程序中显示的表的内容。通过向测试或组件中添加表检查点,可以检查表的单元格中是否显示了指定的值。对于 ActiveX 表,还可以检查表对象的属性。要添加表检查点,可使用"检查点属性"对话框。

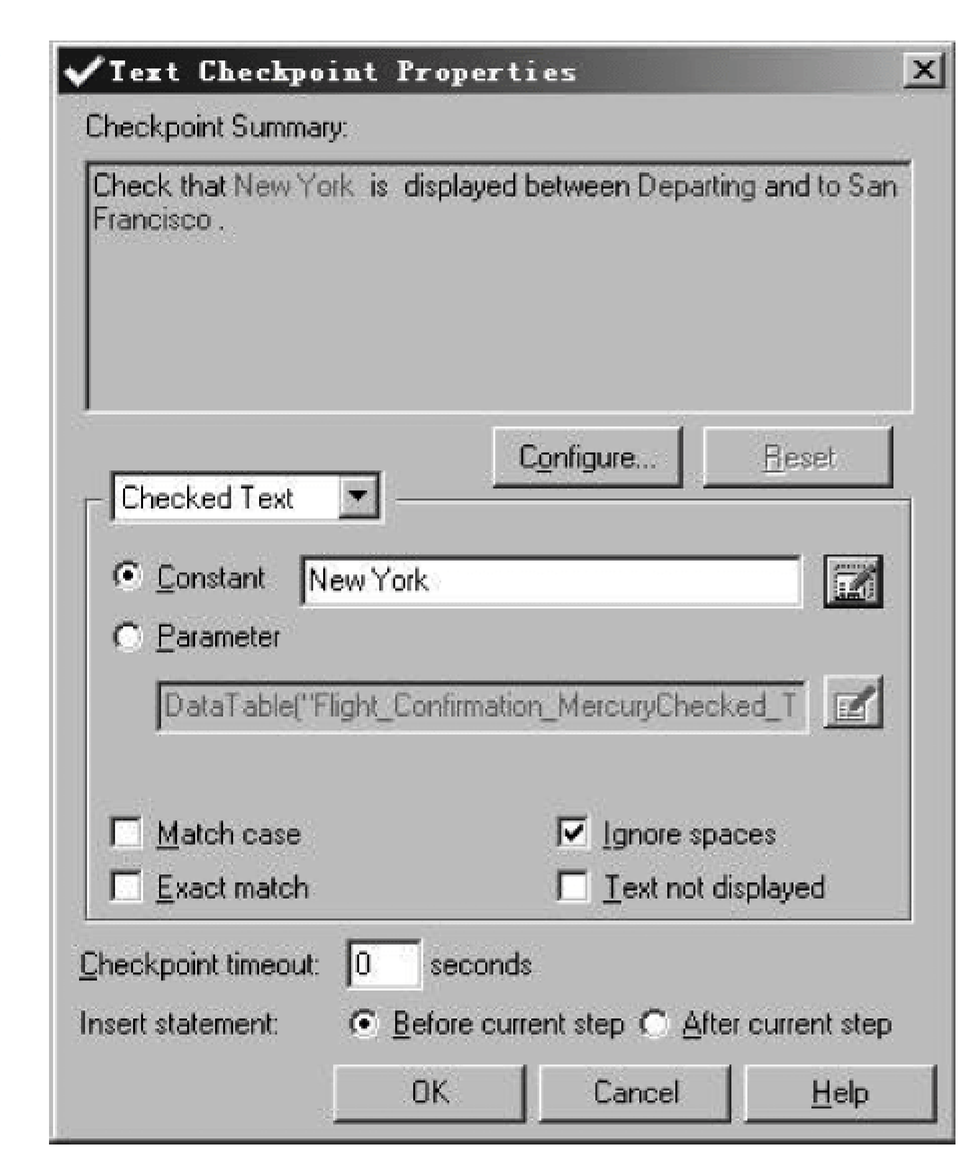


图 8-15 Text Checkpoint Properties 对话框

前面已经添加了标准、网页、文字检查点,接下来在 Checkpoint 测试脚本中再添加一个表格检查点,检查 Book a Flight: Mercury 网页上航班的价格。

创建表格检查点的步骤如下。

(1) 选取要建立检查点的网页。

展开 Action1→Welcome: Mercury Tours 选择 Book a Flight: Mercury 页面,在 Active Screen 中会显示相应的页面。

(2) 建立表格检查点。

在 Active Screen 中,在第一个航班的价钱 270 上单击鼠标右键,选择 Insert Standard Checkpoint 选项打开 Object Selection-Checkpoint Properties 对话框,如图 8-16 所示。

刚打开时选取的是 WebElement:270,这时要选择上一层的 WebTable 对象,在这个例子中选择 WebTable: New York to San Francisco。单击 OK 按钮打开 Table Checkpoint Properties 对话框,显示整个表格的内容,如图 8-17 所示。

预设每一个字段都会被选择,表示所有字段都会检查,可以对某个字段双击,取消检查字段,或者选择整个栏和列,执行选取或取消的动作。

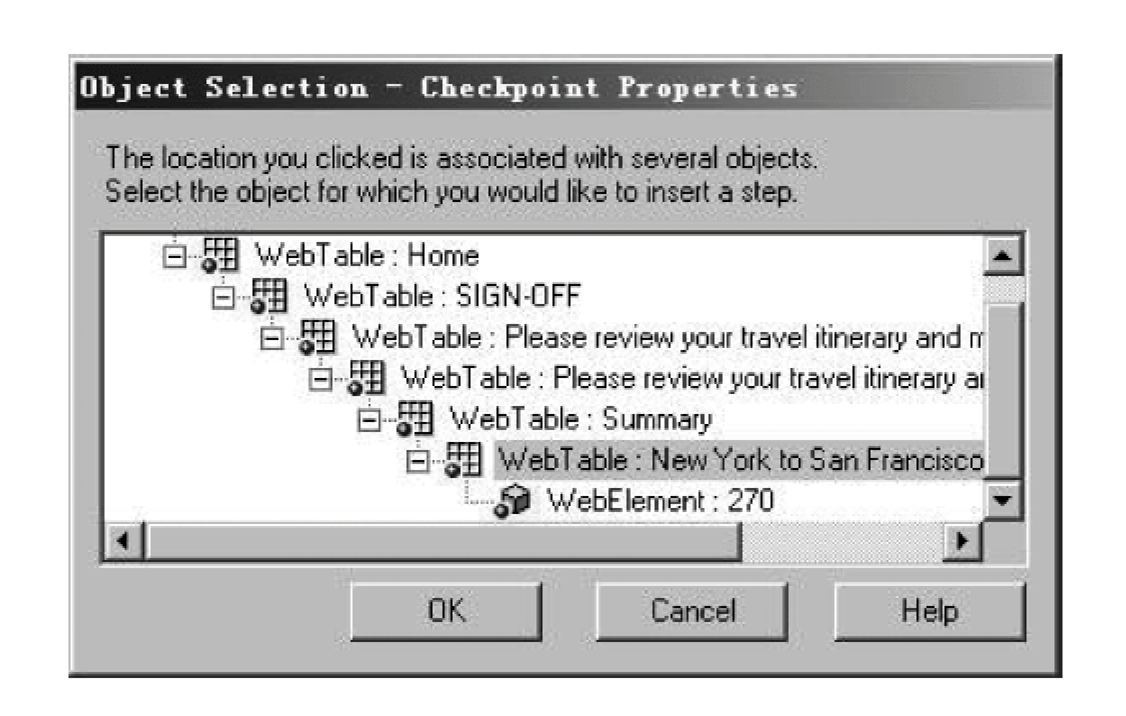


图 8-16 Object Selection-Checkpoint Properties 对话框

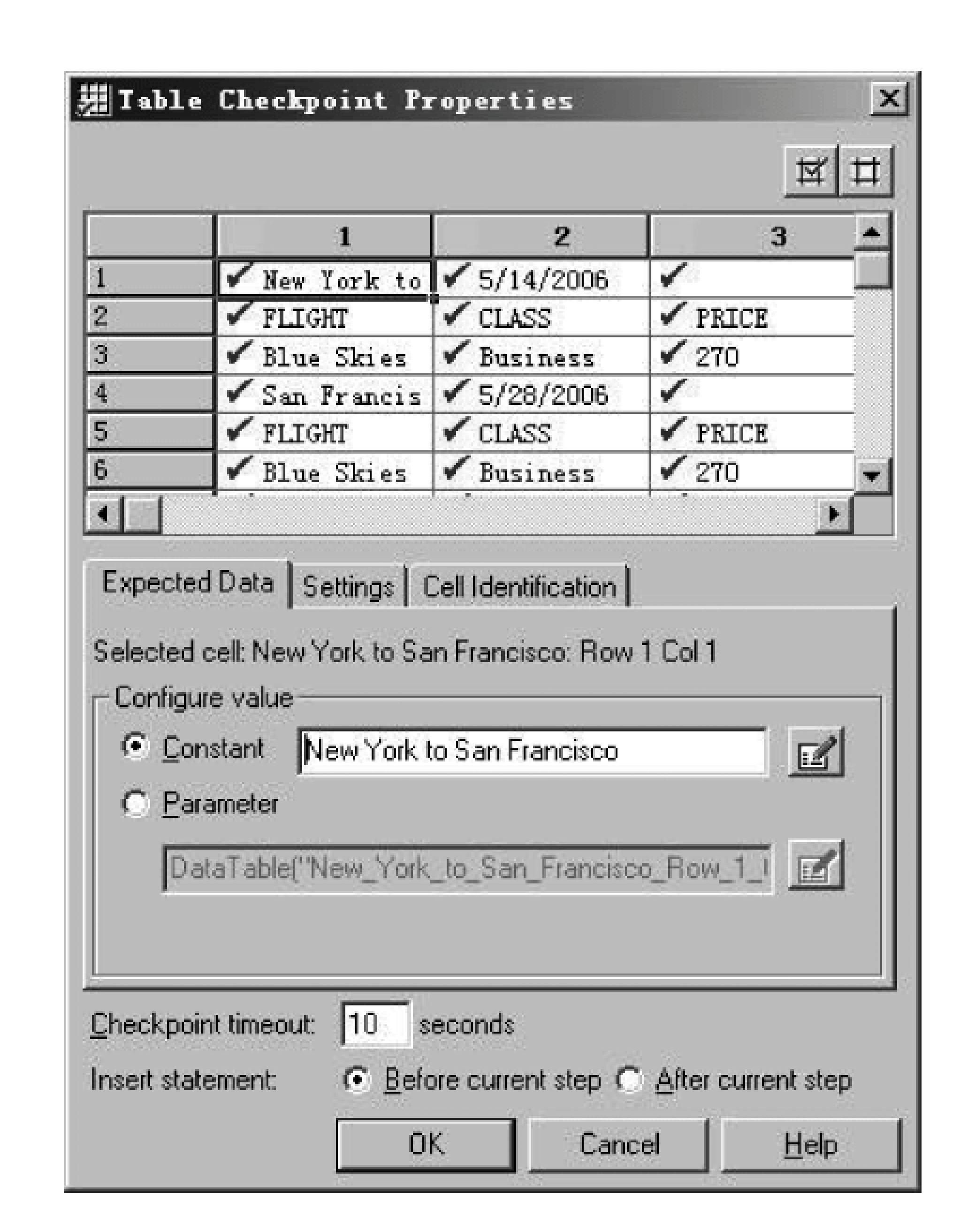


图 8-17 Table Checkpoint Properties 对话框

在每个字段的列标题上双击，取消勾选的图标，然后在 270 字段处双击，这样执行时 QuickTest 只会对这个字段值做检查，如图 8-18 所示。

	1	2	3
1	New York to S	5/14/2006	
2	FLIGHT	CLASS	PRICE
3	Blue Skies Ai	Business	✓ 270
4	San Francisco	5/28/2006	
5	FLIGHT	CLASS	PRICE
6	Blue Skies Ai	Business	270

图 8-18 字段值检查

(3) 单击 OK 按钮关闭对话框。

QuickTest 会在测试脚本中，Book a Flight: Mercury 页面下加上一个表格检查点。

(4) 在工具栏上单击 Save 按钮保存脚本。

3. 执行并分析使用检查点的测试脚本

前面在脚本中添加了 4 个检查点，现在，运行 Checkpoint 测试脚本，分析插入检查点后脚本的运行情况。

(1) 在工具栏上单击 Run 按钮，弹出如图 8-19 所示的对话框。

这个页面是询问将本次测试结果保存在哪个目录，选择 New run results folder 单选按钮，接受默认设置，单击 OK 按钮确认。这时 QuickTest 会按照脚本中的操作，一步一步进行测试，操作过程和手工操作是完全一样的。

(2) 当 QuickTest 执行完测试脚本后，测试执行结果窗口会自动开启。如果所有的检查点都通过了验证，运行结果为 Passed。如果有一个或多个检查点没有通过验证，则运行结果显示为 Failed，如图 8-20 所示。

在图 8-20 中可以看到，设置的 4 个检查点都通过了验证，下面看一下各个检查点的验证结果。

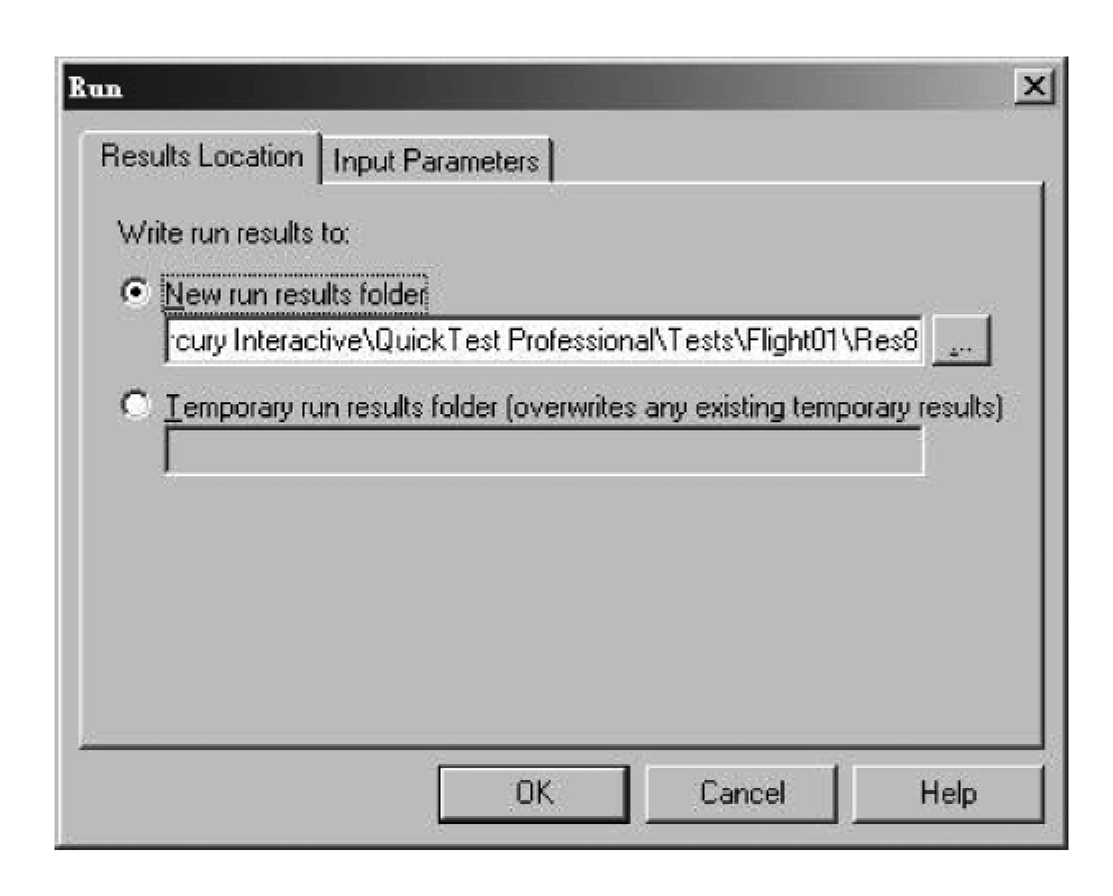


图 8-19　Run 对话框

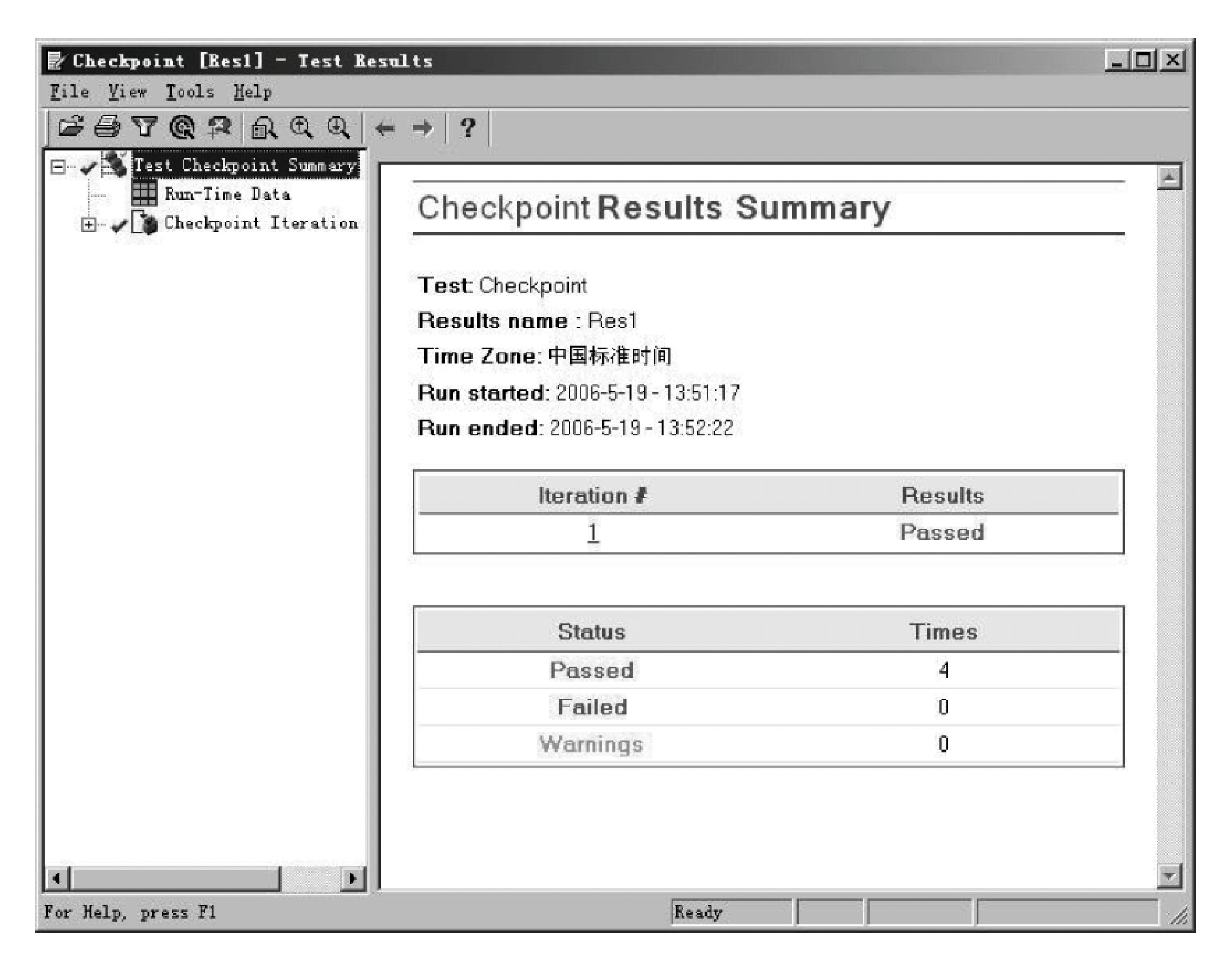


图 8-20　测试运行结果

1）验证网页检查点

在 Test results tree 中展开 Checkpoint Iteration 1（Row 1）→Action1 Summary→Welcome：Mercury Tours→Book a Flight：Mercury，并选择 Checkpoint"Book a Flight：Mercury"。

在右边的 Details 窗口中，可以看到网页检查点的详细信息。例如，网页检查点检查了哪些项目，如图 8-21 所示。

由于所有网页检查的项目，其实际值与预期值相符，所以这个网页检查点的结果为 Passed。

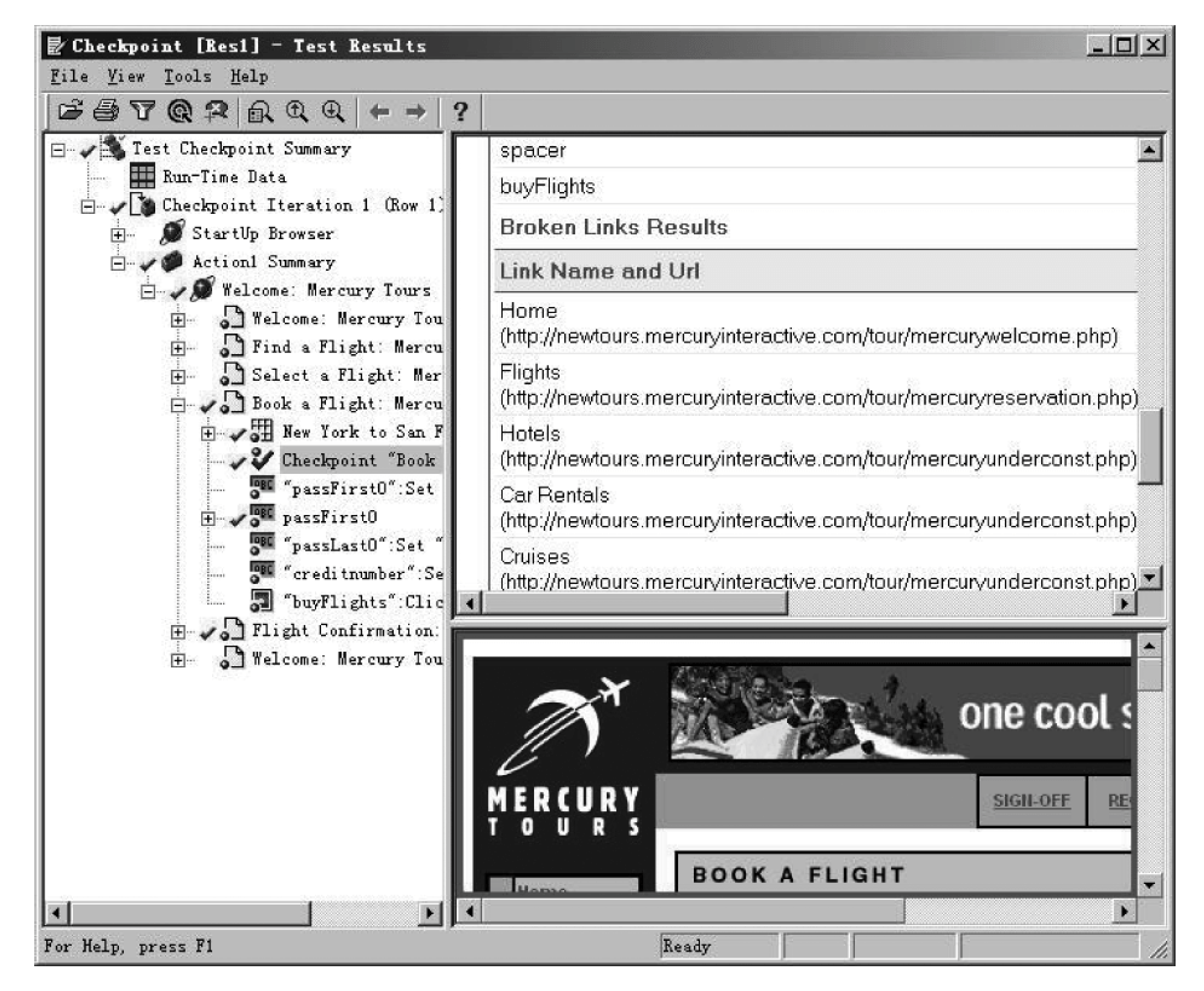


图 8-21 网页检查点的详细信息

2）验证表格检查点

在 Test results tree 中展开 Book a Flight：Mercury→New York to San Francisco，并选择 Checkpoint"New York to San Francisco"。

在 Details 窗口中可以看到表格的详细结果，也可以在窗口下方看到整个表格的内容，被检查的字段以黑色的粗体文字显示，没有检查的字段以灰色文字显示，如图 8-22 所示。

这个表格检查点检查的字段值，其实际值与预期值相符，所以检查点的结果为 Passed。

3）验证标准检查点

在 Test results tree 中展开 Book a Flight：Mercury→passFirst0，并选择 Checkpoint"passFirst0"。

在 Details 窗口中可以看到标准检查点的详细结果，如检查了哪些属性，以及属性的值，如图 8-23 所示。

4）验证文字检查点

在 Test results tree 中展开 Checkpoint Iteration 1（Row 1）→Action1 Summary→Welcome：Mercury Tours→Flight Confirmation：Mercury，并选择 Checkpoint"New York"。显示如图 8-24 所示界面，因为文字检查点的实际值与预期值相同，所以检查点的结果为 Passed。

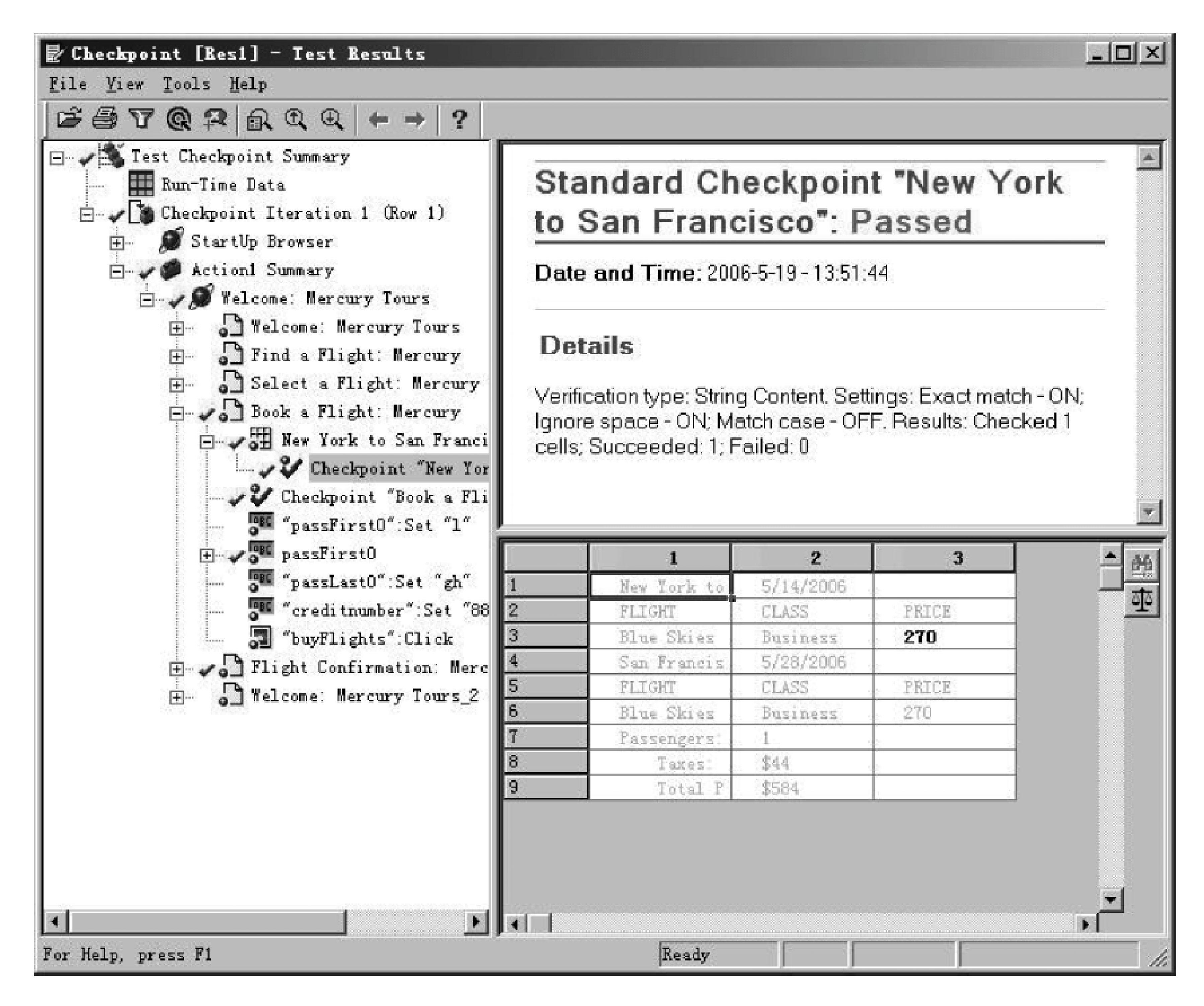


图 8-22　表格的详细结果

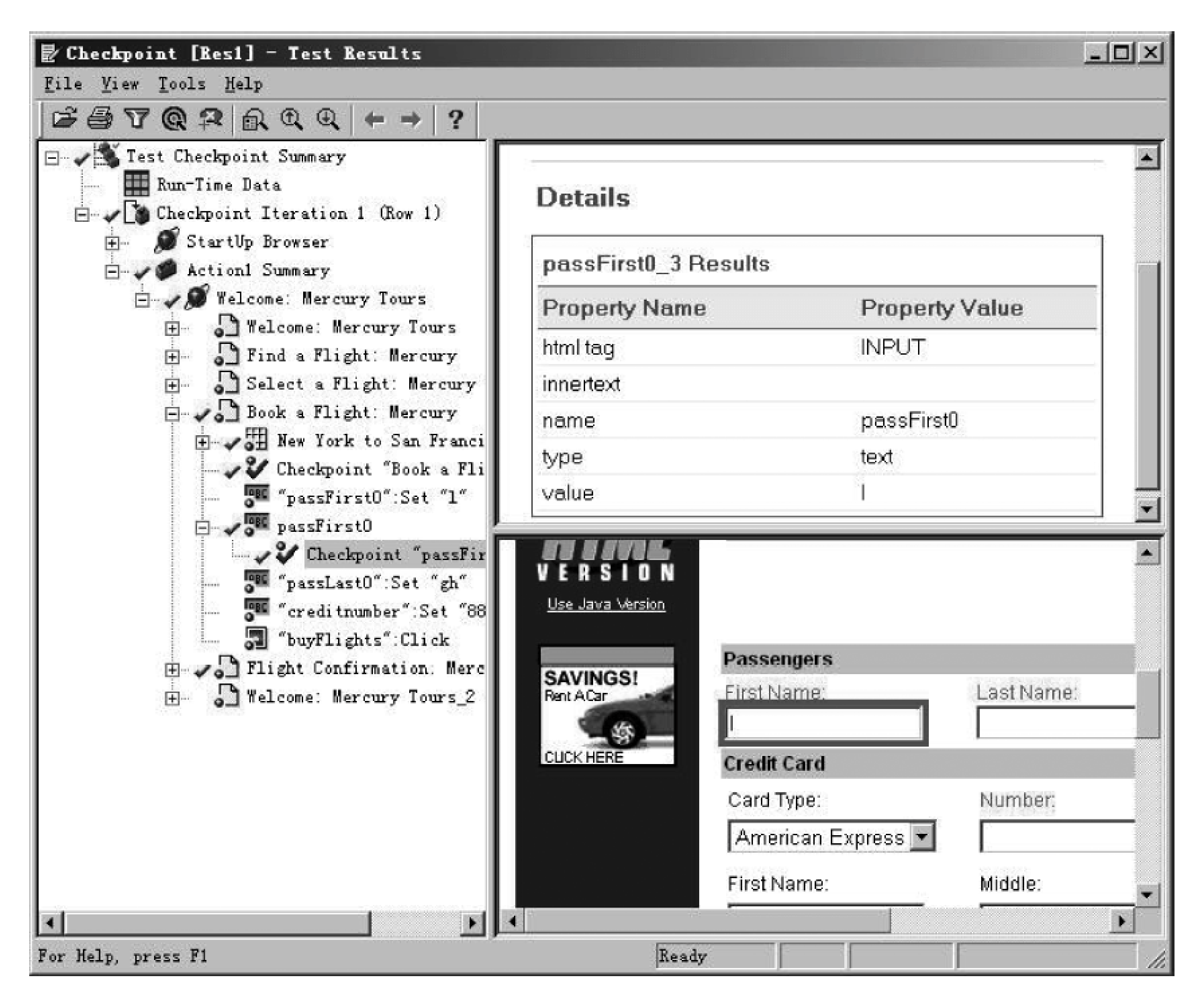


图 8-23　标准检查点的详细结果

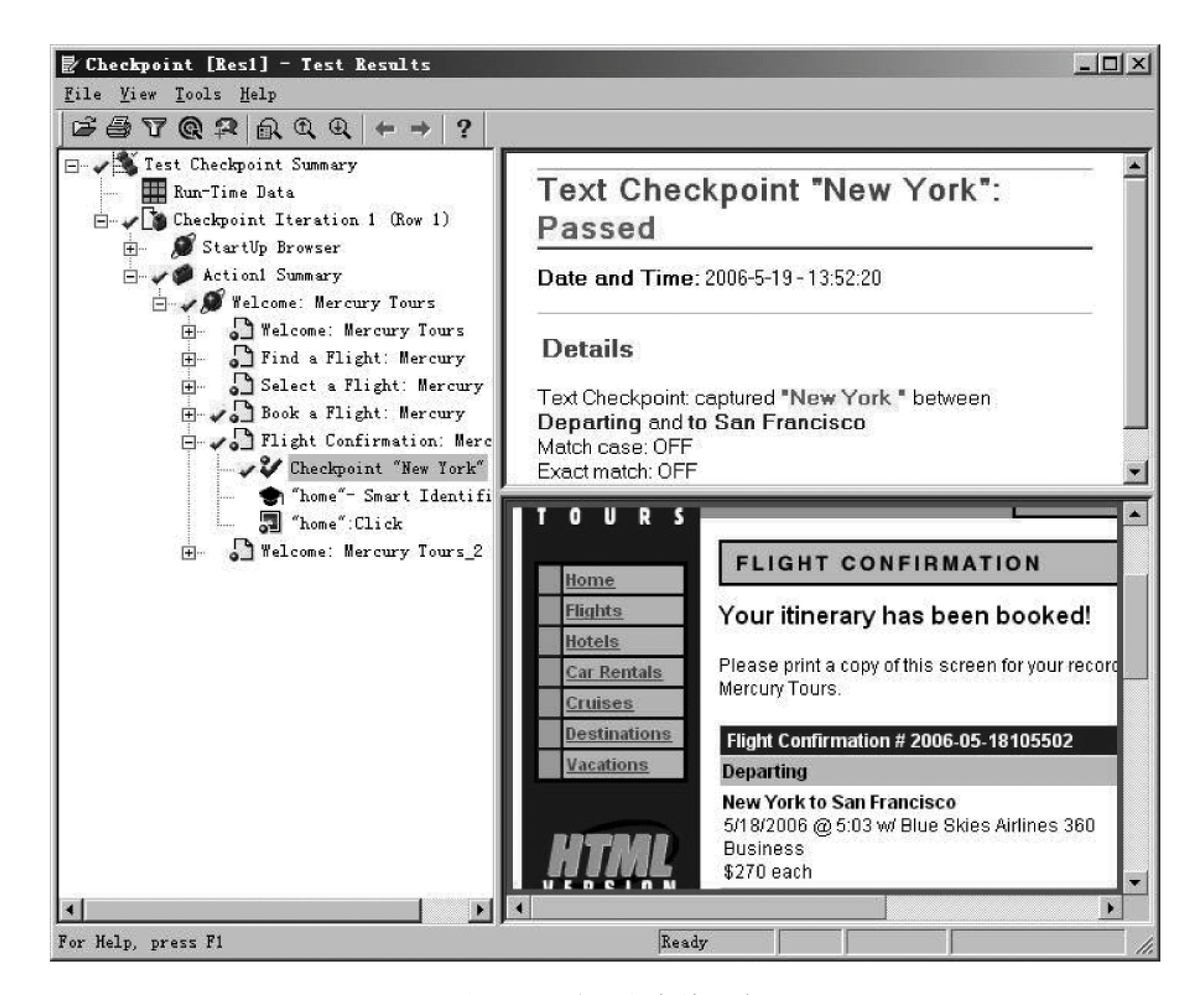


图 8-24　验证文字检查点

8.1.4　参数化

在测试应用程序时，可能想检查对应用程序使用不同输入数据进行同一操作时，程序是否能正常地工作。在这种情况下，可以将这个操作重复录制多次，每次输入不同的数据，这种方法虽然能够解决问题，但实现起来太笨拙了。QuickTest 提供了一个更好的方法来解决这个问题——参数化测试脚本。参数化测试脚本包括数据输入的参数化和检测点的参数化。

使用 QuickTest 可以通过将固定值替换为参数，扩展基本测试或组件的范围。该过程（称为参数化）大大提高了测试或组件的功能和灵活性。

可在 QuickTest 中使用参数功能，通过参数化测试或组件所使用的值来增强测试或组件。参数是一种从外部数据源或生成器赋值的变量。

QuickTest 可以参数化测试或组件中的步骤和检查点中的值，还可以参数化操作参数的值。如果希望参数化测试或组件中多个步骤中的同一个值，可能需要考虑使用数据驱动器，而不是手动添加参数。

1. 参数化步骤和检查点中的值

录制或编辑测试或组件时，可以参数化步骤和检查点中的值，可以参数化选定步骤的对象属性的值，还可以参数化为该步骤定义的操作（方法或函数参数）的值。

例如，应用程序或网站可能包含一个带有编辑字段的表单，用户可以在该编辑字段中输入用户名。测试者可能希望测试应用程序或网站是否读取该信息并将其正确显示在对话框中。可以插入一个对已登录的用户名使用内置环境变量的文本检查点，以检查显示的信息是否正确。

通过参数化检查点属性的值，可以检查应用程序或网站如何基于不同的数据执行相同的操作。

例如，如果要测试 Mercury Tours 示例网站，可以创建一个检查点，以便检查预订机票后该机票是否被正确预订。假设需要检查针对各种不同目的地所预订的航班是否正确，可以为目的地信息添加一个数据表参数，而不是为每个目的地分别创建带有单独检查点的不同测试或组件。对于测试或组件的每次循环，QuickTest 都会针对不同目的地检查航班信息。

1）参数化对象和检查点的属性值

可以在“对象属性”或“对象库”对话框中参数化对象的一个或多个属性的值。可以在“检查点属性”对话框中参数化检查点的一个或多个属性的值。

采用下列方式可以打开“对象属性”对话框或“检查点属性”对话框。

（1）选择“步骤”→“对象属性”，或者右键单击某个步骤并选择“对象属性”选项。将打开“对象属性”对话框。

（2）选择“工具”→“对象库”，单击“对象库”工具栏按钮，或者右键单击包含该对象的操作或组件，然后选择“对象库”选项。将打开“对象库”对话框。

（3）选择“步骤”→“检查点属性”，或者右键单击该检查点并选择“检查点属性”选项。然后在对话框的“配置值”区域中选择参数，如图 8-25 所示。

如果该值已经参数化，则 Parameter 框中将显示该值的当前参数定义。如果该值尚未参数化，则 Parameter 框中将显示该值的默认参数定义。

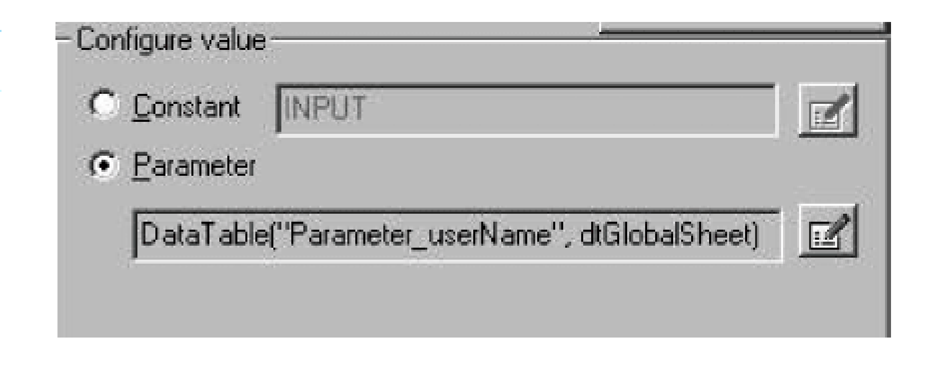


图 8-25　“配置值”区域

2）参数化操作的值

如果步骤中使用的方法或函数具有参数，则可以根据需要参数化该参数值。例如，如果操作使用 Click 方法，则可以参数化 x 参数、y 参数或这两者的值。

在关键字视图中选择已参数化的值时，将显示该参数类型的图标。例如，在图 8-26 的片段中，已将 Set 方法的值定义为随机数字参数。每次运行测试或组件时，QuickTest 都会在 creditnumber 编辑框中输入一个随机数字值。



图 8-26　参数化操作的值

可以使用视图中的“值”列中的参数化图标来参数化操作值。

单击参数化图标，打开“值配置选项”对话框，将显示当前定义的值，如图 8-27 所示。

选择“参数”。如果该值已经参数化，则“参数”部分将显示该值的当前参数定义。如果

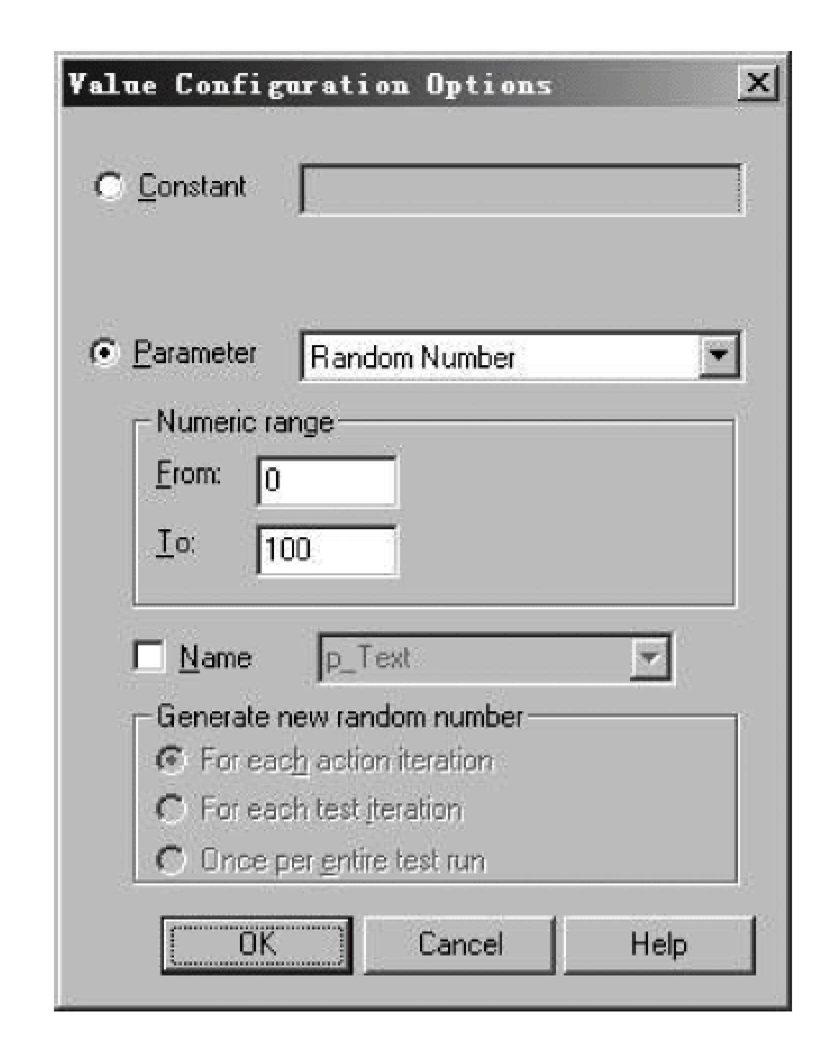


图 8-27 “值配置选项”对话框

该值尚未参数化，则“参数”部分将显示该值的默认参数定义。单击 OK 按钮接受显示的参数语句并关闭该对话框。

选择一个尚未参数化的值时，QuickTest 会为该值生成默认参数定义。表 8-4 描述了如何确定默认参数设置。

表 8-4 确定默认参数设置

执行参数化时	条　　件	默认参数类型	默认参数名
操作中的步骤或检查点的值	至少在当前操作中定义了一个输入操作参数	操作参数	在“操作属性”对话框的“参数”选项卡中显示第一个输入参数
嵌套操作的输入操作参数值	至少为调用该嵌套操作的操作定义了一个输入操作参数	操作参数	在调用操作的“操作属性”对话框的“参数”选项卡中显示第一个输入参数
顶层操作调用的输入操作参数值	至少为测试定义了一个输入参数	测试参数	在“测试设置”对话框的“参数”选项卡中显示第一个输入参数
组件中的步骤或检查点的值	至少为该组件定义了一个输入参数	组件参数	在“业务组件设置”对话框的“参数”选项卡中显示第一个输入参数

如果上述相关条件不为真，则默认参数类型为“数据表”。如果接受了默认参数详细信息，QuickTest 将用基于选定值的名称新建一个数据表参数。

2．参数种类

QuickTest 有以下 4 种类型的参数。

(1) 测试、操作或组件参数，通过它可以使用从测试或组件中传递的值，或者来自测试中的其他操作的值。为了在特定操作内使用某个值，必须将该值通过测试的操作层次结构向下传递到所需的操作。然后，可以使用该参数值来参数化测试或组件中的步骤。例如，假设要使用从运行(调用)测试的外部应用程序传递到测试中的某个值来参数化 Action3 中的一个步骤。可将该值从测试级别传递到 Action1 (顶层操作)至 Action3 (Action1 的子操作)，然后使用该操作输入参数值(从外部应用程序传递的值)来参数化所需的步骤。

（2）数据表参数，通过它可以创建使用测试者所提供的数据多次运行的数据驱动的测试（或操作）。在每次重复（或循环）中，QuickTest 均使用数据表中不同的值。例如，假设应用程序或网站包含一项功能，用户可以通过该功能从成员数据库中搜索联系信息。当用户输入某个成员的姓名时，将显示该成员的联系信息，以及一个标记为“查看<MemName>的照片”的按钮，其中，<MemName>是该成员的姓名。可以参数化按钮的名称属性，以便在运行会话的每次循环期间，QuickTest 可标识不同的照片按钮。

（3）环境变量参数，通过它可以在运行会话期间使用来自其他来源的变量值。这些变量值可能是测试者所提供的值，或者是 QuickTest 基于测试者选择的条件和选项而生成的值。例如，可以让 QuickTest 从某个外部文件读取用于填写 Web 表单的所有值，或者可以使用 QuickTest 的内置环境变量之一来插入有关运行测试或组件的计算机的当前信息。

（4）随机数字参数，通过它可以插入随机数字作为测试或组件的值。例如，要检查应用程序处理大小机票订单的方式，可以让 QuickTest 生成一个随机数字，然后将其插入到“票数”编辑字段中。

1）使用数据表参数

可以通过创建数据表参数来为参数提供可能的值列表。通过数据表参数可以创建测试者所提供的数据多次运行的数据驱动测试、组件或操作。在每次重复中，QuickTest 均使用数据表中不同的值。

例如，考虑 Mercury Tours 示例网站，通过该网站可预订航班请求。要预订航班，需要提供航班路线，然后单击“继续”按钮。如图 8-28 所示该网站将针对请求的路线返回可用的航班。

A1　New York

	Departure	arrival	C	D	E	F	G
1	New York	San Francisco					
2	London	Paris					
3	Frankfure	Acapulco					
4							
5							
6							
7							
8							

Global　Action1

图 8-28　针对一条路线来检查可用航班

可通过访问网站并录制大量查询的提交来执行该测试。这是一个既费时又费力的低效解决方案。通过使用数据表参数，可以连续对多个查询运行测试或组件。

参数化测试或组件时，需要首先录制访问网站并针对所请求的一条路线来检查可用航班的步骤。然后将录制的路线替换为某个数据表参数，并在数据表的全局表中添加自己的数据集，每条路线一个。

新建数据表参数时，将在数据表中添加新的一列，并将参数化的当前值放在第一行中。如果要对值进行参数化并选择现有的数据表参数，则将保留所选参数的列中的值，并且这些值不会被参数的当前值覆盖。

如图 8-29 所示，表中的每个列都表示单个数据表参数的值列表。列标题是参数名。表中的每一行都表示 QuickTest 在测试或组件的单次循环期间为所有参数提交的一组值。运行测试或组件时，QuickTest 将针对表中的每一行数据运行一次测试或组件循环。例如，如果测试在数据表的全局表中有 10 行，则运行 10 次循环。

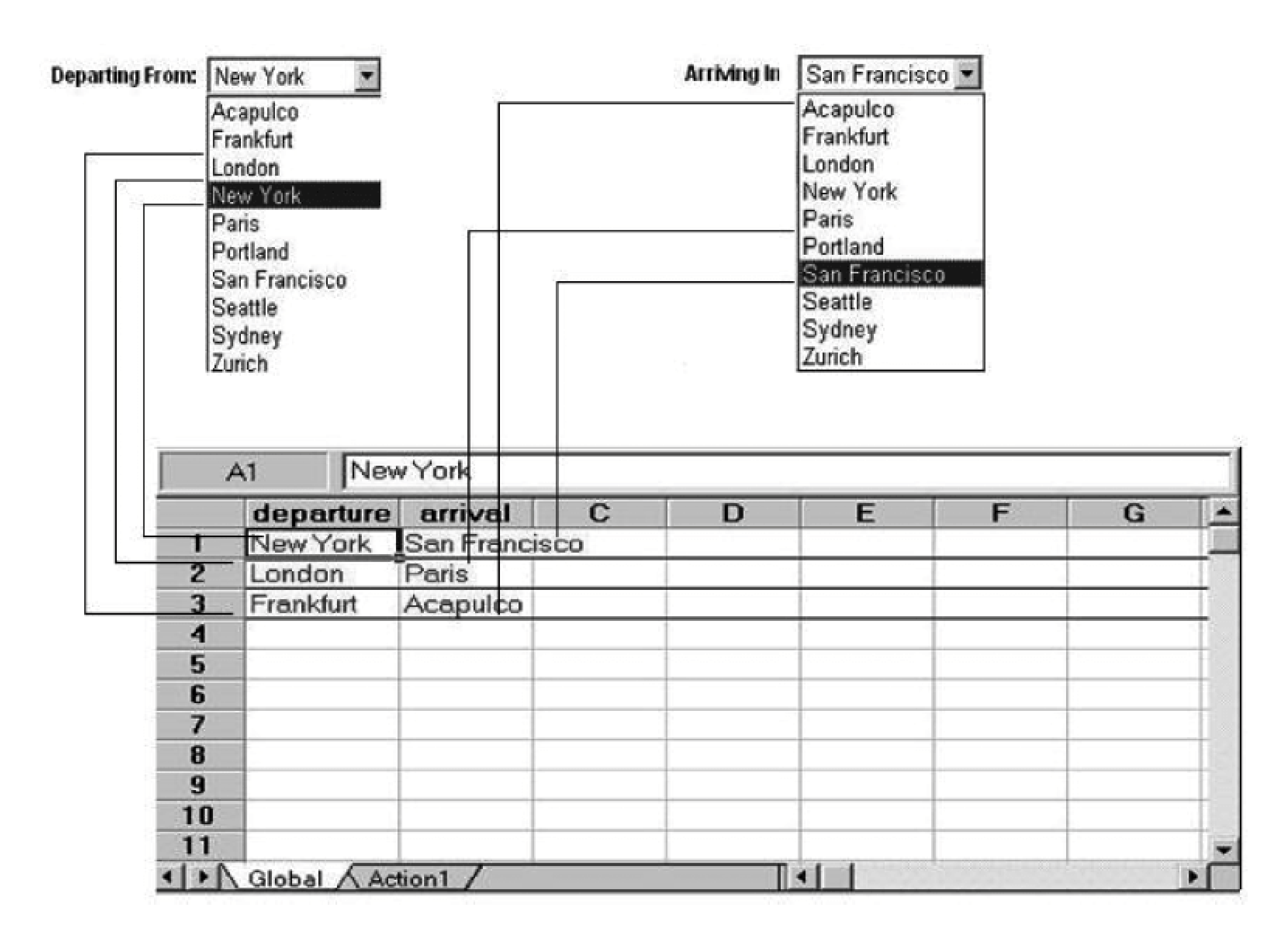


图 8-29 测试循环举例

在上面的例子中，当运行测试时，QuickTest 将为每一个路线分别提交一个查询。

2）使用环境变量参数

QuickTest 可以插入环境变量列表中的值，该列表是可通过测试访问的变量和相应值的列表。在测试运行的整个过程中，无论循环次数是多少，环境变量的值始终保持不变，除非在脚本中以编程方式更改变量的值。

QuickTest 有以下三种环境变量：用户定义的内部环境变量、用户定义的外部环境变量以及内置环境变量。

用户定义的内部环境变量——在测试内定义的变量。这些变量与测试一起保存，并且只能在定义这些变量的测试内访问。在“测试设置”对话框或“参数选项”对话框的“环境”选项卡中，可以创建或修改测试中用户定义的内部环境变量。

用户定义的外部环境变量——在活动外部环境变量文件中预定义的变量。可根据需要创建任意多的文件，并为每个测试选择一个适当的文件，或者更改用于每个测试运行的文件。

内置环境变量——表示有关测试和运行测试的计算机的信息的变量，例如测试路径和操作系统。从所有测试和组件中都可以访问这些变量，并且它们都被指定为只读变量。

3）使用随机数字参数

当选择“随机数字”作为参数类型时，可以通过“参数选项”对话框将参数配置为使用随机数字。“值配置选项”对话框的“参数”部分与“参数选项”对话框非常相似，如图 8-30 所示。

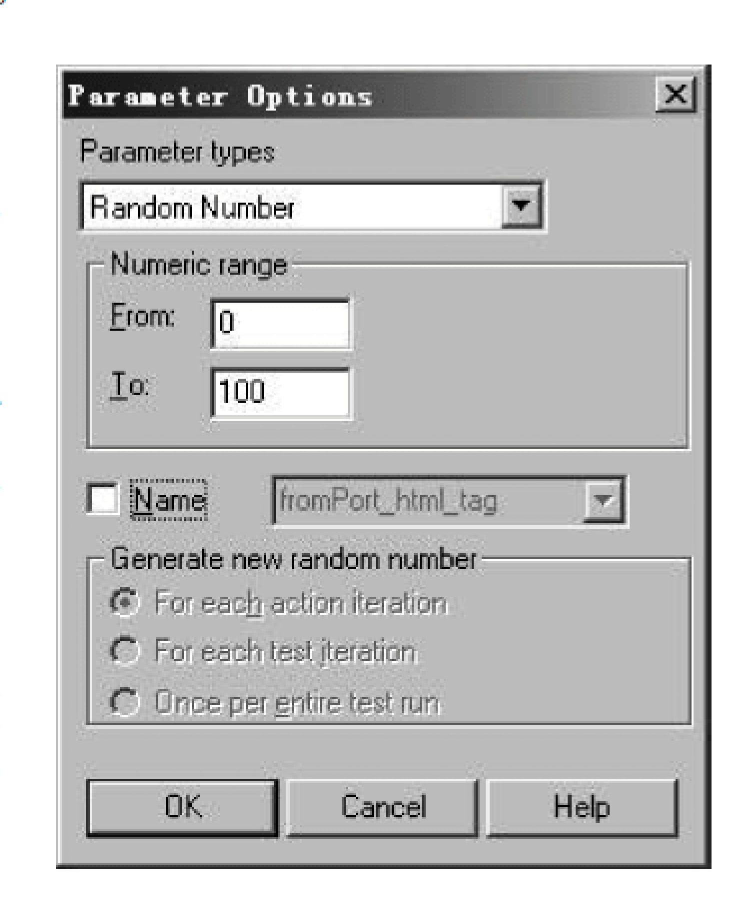


图 8-30 “参数选项”对话框

数字范围——指定用于生成随机数字的范围。默认情况下，随机数字范围介于 0～100 之间。可通过在“从”和“到”框中输入不同的值来修改此范围。该范围必须介于 0～2 147 483 647（包含）之间。

名称——指定参数的名称。通过为随机参数指定名称可以在测试中多次使用同一个参数。可以选择现有的命名参数，或者通过输入新的描述性名称来新建命名参数。

生成新随机数字——定义命名随机参数的生成计时。选中“名称”复选框时会启用该框。可以选择下列选项之一。

（1）为每次操作循环：在每次操作循环结束时生成一个新数字。

（2）为每次测试循环：在每次全局循环结束时生成一个新数字。

（3）为整个测试运行生成一次：第一次使用参数时生成一个新数字。在整个测试运行中，对参数使用同一个数字。

3. 参数化测试脚本

1）定义参数

前面学习了参数的种类以及参数化步骤和检查点中的值，现在使用 Checkpoint 脚本，在测试脚本中，New York 是个常数值，也就是说，每次执行测试脚本预订机票时，出发地点都在 New York。现在，将测试脚本中的出发地点参数化，这样，执行测试脚本时就会以不同的出发地点去预订机票了。

（1）打开 Checkpoint 测试脚本，将脚本另存为 Parameter，然后选择要参数化的文字：在视图树中展开 Action1→Welcome：Mercury Tours→Find a Flight：Mercury。

（2）在视图树中选择 fromPort 右边的 Value 字段，然后再单击参数化图标，开启 Value Configuration Options 对话框，如图 8-31 所示。

（3）设置要参数化的属性，选择 Parameter 选项，如图 8-32 所示，这样就可以用参数值来取代“New York”这个常数。在参数中选择 DataTable 选项，这样这个参数就可以从 QuickTest 的 DataTable 中取得，将参数的名字改为“departure”。

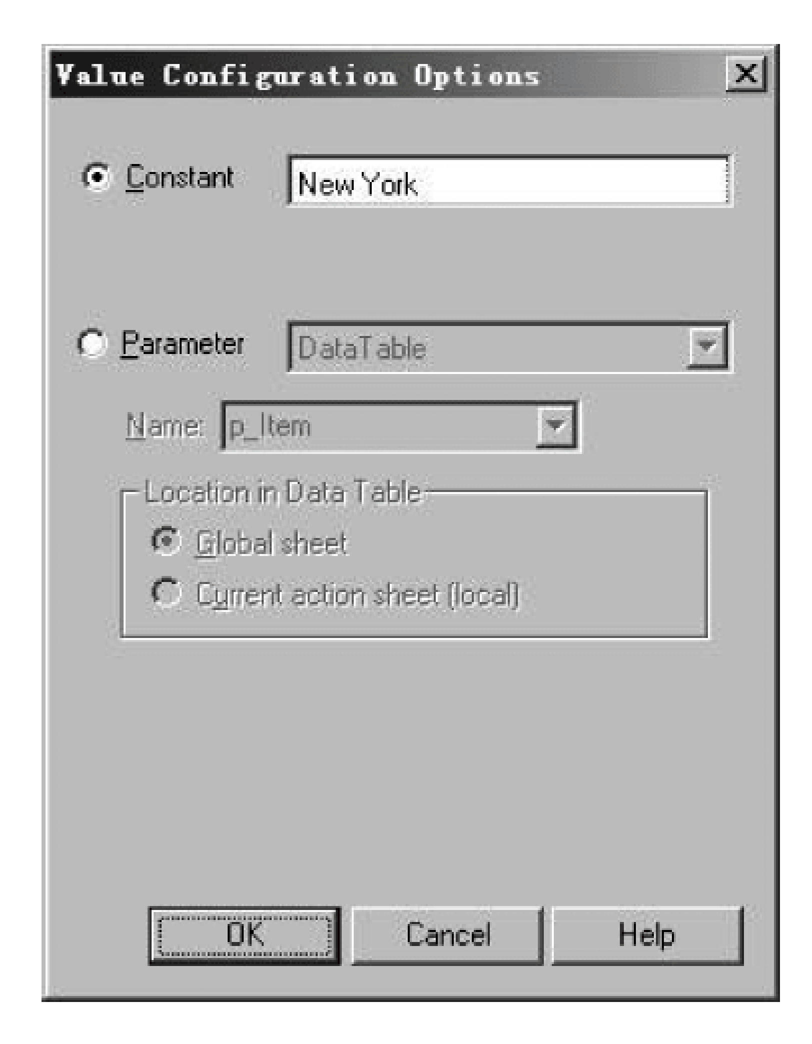


图 8-31　Value Configuration Options 对话框

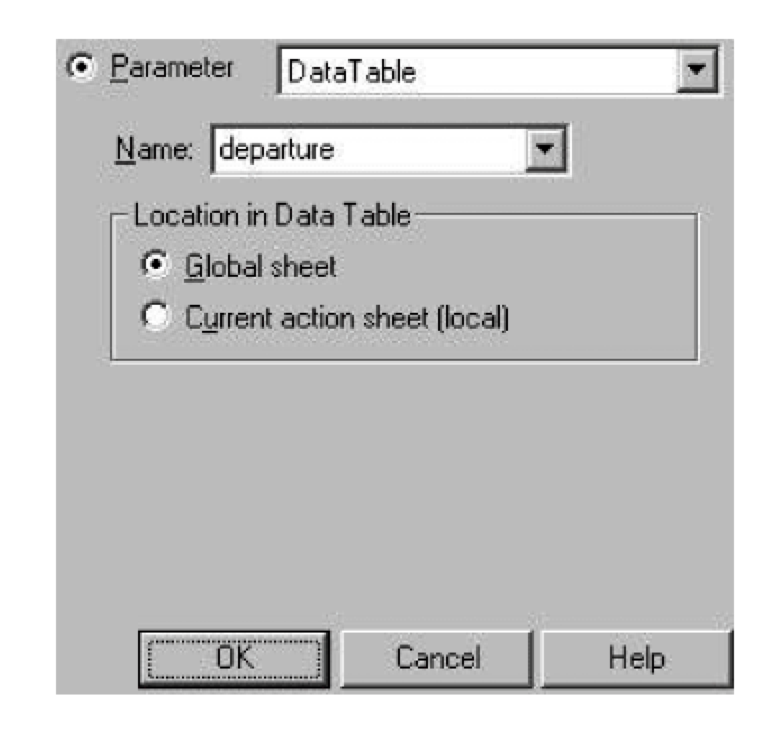


图 8-32　设置要参数化的属性

(4) 单击 OK 按钮确认，QuickTest 会在 DataTable 中新增 departure 参数字段，并且插入一行 New York 的值，“New York”会成为测试脚本执行使用的第一个值，如图 8-33 所示。

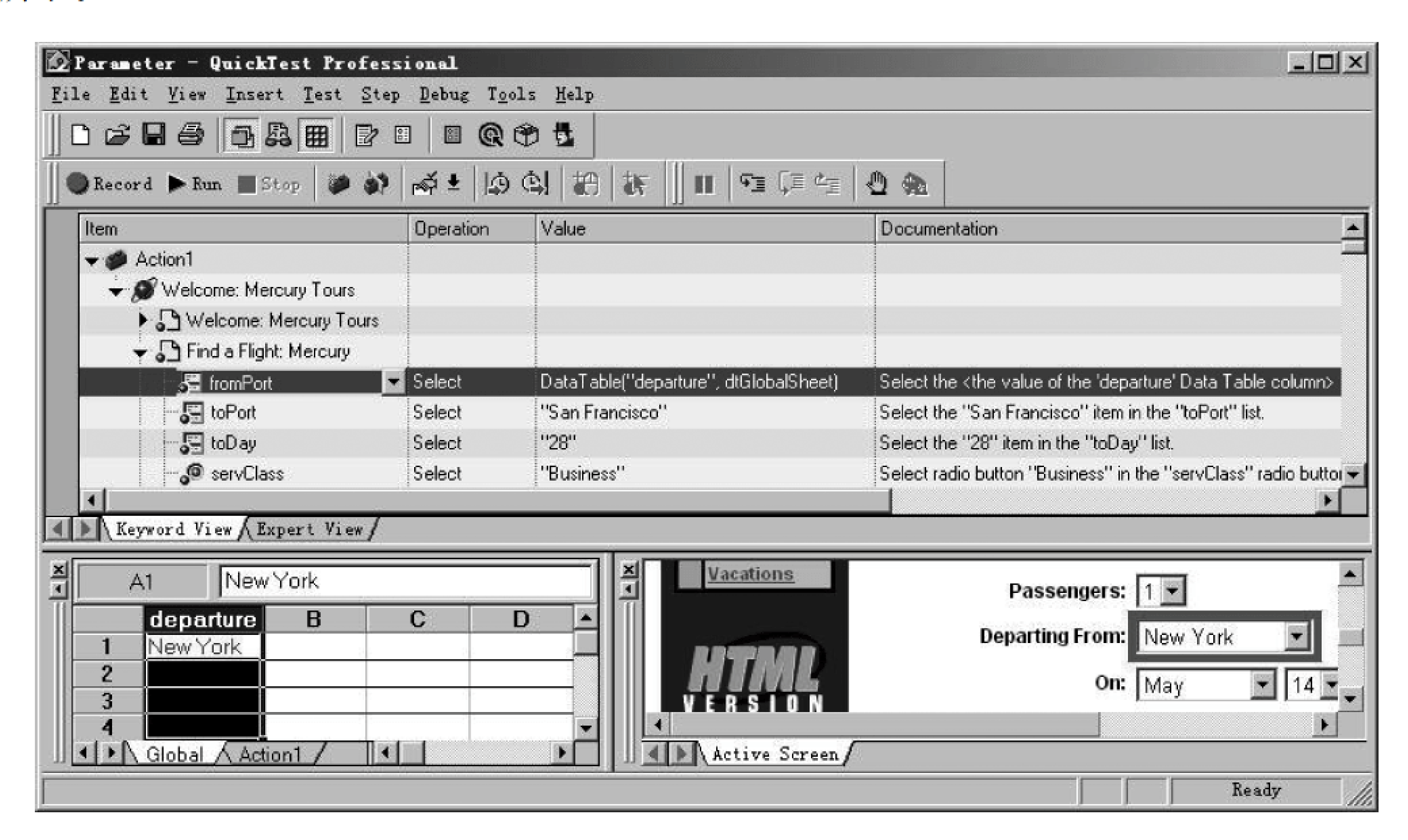


图 8-33 新增 departure 参数字段

参数化以后可以看到树视图中的变化，在参数之前，这个测试步骤显示“fromPort …Select… New York”，现在，这个步骤变成了“fromPort …Select… Data Table ("departure",dtGlobalSheet)”。而且当单击 Value 字段时，Value 字段会显示为 <departure>，表示此测试步骤已经被参数化，而且其值从 DataTable 中的 departure 字段中获得。

(5) 在 departure 字段中加入出发点资料，使 QuickTest 可以使用这些资料执行脚本。

在 departure 字段的第二行、第三行分别输入：Portland、Seattle。

(6) 保存测试脚本。

2) 修正受到参数化影响的步骤

当测试步骤被参数化以后，有可能会影响到其他的测试步骤也要参数化。例如，为了验证在 Flight Confirmation 网页中是否出现“New York”，在网页上添加了一个文字检查点。那么，就要对出发地的文字检查点做参数化，以符合对出发地点参数化的预期结果。

修正文字检查点，首先在树视图中，展开 Action1→Welcome：Mercury Tours→Flight Confirmation：Mercury 页面，然后单击鼠标右键，选择 Checkpoint Properties 选项，打开 Text Checkpoint Properties 对话框，如图 8-34 所示。

在 Checked Text 的 Constant 字段中显示为“New York”，表示测试脚本在每次执行时，这个文字检查点的预期值都为“New York”。选择 Parameter，单击旁边的 Parameter Options 按钮，打开 Parameter Options 对话框，如图 8-35 所示。

在参数类型选择框中选择 DataTable 选项，在名字选择框中选择 departure 选项，指明这个文字检查点使用 departure 字段中的值当成检查点的预期值。




图 8-34　Text Checkpoint Properties 对话框

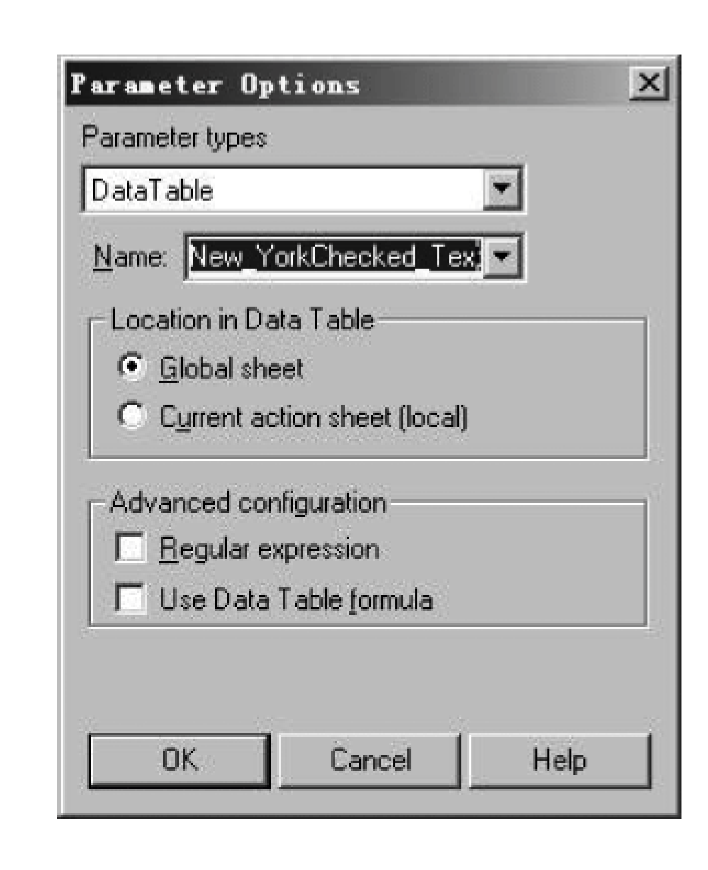


图 8-35　Parameter Options 对话框

单击 OK 按钮关闭对话框，这样文字检查点也被参数化了。

3）执行并分析使用参数的测试脚本

参数化测试脚本后，运行 Parameter 测试脚本。QuickTest 会使用 DataTable 中的 departure 字段值，执行三次测试脚本。

执行测试脚本：单击工具栏上的 Run 按钮，开启 Run 对话框，选取 New run results folder，其余为默认值，单击 OK 按钮开始执行脚本。当脚本运行结束后，会开启测试结果窗口。

在树视图中，展开 Parameter Iteration2→Action1 Summary→Welcome Mercury Tours→Flight Confirmation：Mercury，选择 Checkpoint"New York"，显示如图 8-36 所示。

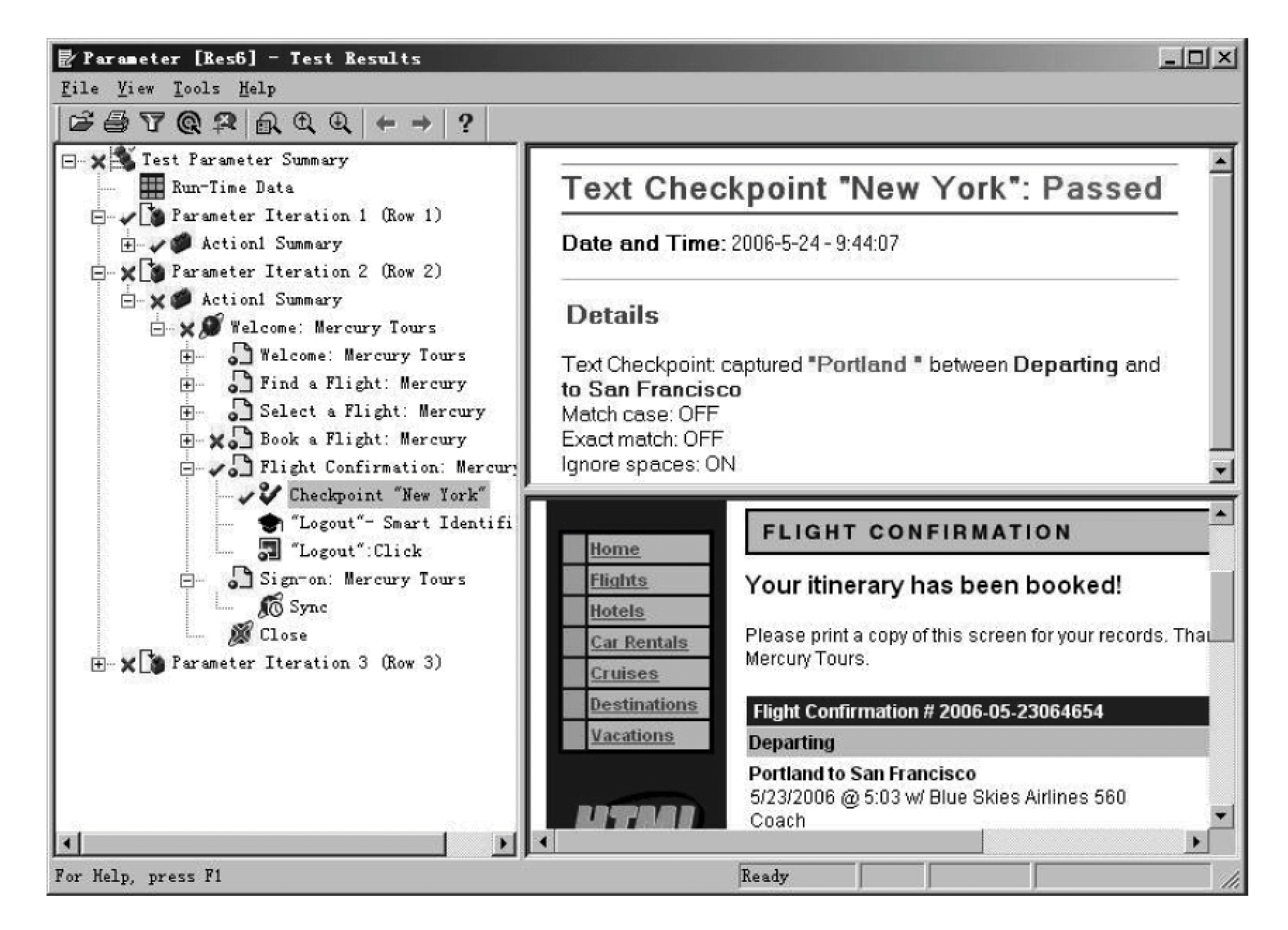


图 8-36　执行测试脚本

在检查点 Details 窗口中，显示 Portland 为预期记录同时也是实际的值，所以文字检查点为通过。同时也可以看到在下方的 Application 窗口中，显示机票的出发地点也是 Portland。

在图 8-36 中可以看出，虽然每次执行时，文字检查点的结果是通过的，但是第二次与第三次的执行结果仍然为失败。这是因为出发地点的改变，造成在表格检查点中的机票价钱改变，导致表格检查点失败。在以后的课程中，将学习修正表格检查点，让 QuickTest 自动更新表格检查点的预期结果，就可以检查正确的票价了。

8.1.5 输出值

通过 QuickTest 可以检索测试或组件中的值，并将这些值作为输出值存储。此后，就可以检索这些值，并在运行会话的不同阶段使用该值作为输入。

输出值是一个步骤，在该步骤中，捕获测试或组件中某个特定点的一个或多个值，并在运行会话持续时间存储这些值。随后，在运行会话中的不同点，可以将这些值作为输入使用。

可以输出任何对象的属性值。还可以从文本字符串、表单元格、数据库和 XML 文档输出值。

创建输出值步骤时，可以确定运行会话持续时间内的值存储在哪里，以及如何使用这些值。运行会话期间，QuickTest 检索指定点的每个值并将其存储在指定位置。以后当运行会话中需要的值时，QuickTest 将从该位置检索值并根据需要来使用。

1. 创建输出值

1）输出值类型

将输出值步骤添加到测试或组件时，首先选择要输出的值的类别，例如，属性值、文本值或 XML 元素值。然后，就可以确定要输出的值以及每个值的存储位置。在 QuickTest 中可以创建以下几个类别的输出值。

（1）标准输出值

可以使用标准输出值来输出大多数对象的属性值。例如，在基于 Web 的应用程序中，一个网页中的链接数可能基于用户在上一页的表单中所做选择的不同而变化。可以在测试中创建一个输出值，来存储页面中的链接数，还可以使用标准输出值来输出表单元格的内容。

（2）文本和文本区输出值

可以使用文本输出值来输出屏幕或网页中显示的文本字符串。创建文本输出值时，可以输出对象文本的一部分，还可以指定要在输出文本之前和之后输出的文本。

可以使用文本区域输出值来输出 Windows Applications 中屏幕已定义区域内显示的文本字符串。例如，假设在测试的应用程序中，想要存储显示在特定步骤之后的任何错误消息的文本。在 if 语句中，查看带有已知标题栏值（例如 Error）的窗口是否存在。如果该窗口存在，则输出该窗口中的文本（假设窗口大小与所有可能的错误消息的大小相同）。

在使用基于 Windows 的应用程序文本输出值时应注意以下事项。

① 在基于 Windows 的应用程序中创建文本或文本区输出值时使用文本识别机制，有

时会检索到不想要的文本信息(例如隐藏文本和带阴影的文本,这些文本会作为同一字符串的多个副本显示)。

② 此外,在不同的运行会话中,文本(和文本区)输出值的表现方式可能不同,具体取决于使用的操作系统版本、已经安装的 Service Pack、安装的其他工具包、应用程序中使用的 API 等。

(3) 数据库输出值

可以使用数据库输出值,基于在数据库上定义的查询的结果(结果集)来输出数据库单元格内容的值。可以从结果集的全部内容中创建输出值,也可以从其中某一部分创建输出值。在运行会话过程中,QuickTest 从数据库中检索当前数据,并根据指定的设置来输出值。

(4) XML 输出值

可以使用 XML 输出值输出 XML 文档中的 XML 元素和属性的值。运行会话完成后,可以在"测试结果"窗口中查看 XML 输出值的概要结果。还可以通过打开"XML 输出值结果"窗口来查看详细结果。例如,假设网页中的某个 XML 文档包含新车的价目表,可以通过选择要输出的相应的 XML 元素值来输出特定汽车的价格。

表 8-5 给出了每种环境支持的输出值类型。

表 8-5　每种环境支持的输出值类型

输出值类别	Web	标准 Windows	VB	ActiveX	其他环境
标准	S	S	S	S	NA
页(标准)	S	NA	NA	NA	NA
表(标准)	S	NA	NA	S	NA
文本	S	S	S	S	NA
文本区	NS	S	S	S	NA
数据库	NS	NA	NA	NA	S(DbTable)
XML	S	NA	NA	NA	XML 文件

* S——支持　NS——不支持　NA——不适用

2) 存储输出值

定义输出值时,可以指定运行会话期间在哪里以及如何存储每个值。可以将值输出到:测试、操作或组件参数,运行时数据表,环境变量。

(1) 将值存储在测试、操作或组件参数中

可以将值输出到操作或组件参数,以便可以在运行会话后面的部分中使用来自运行会话某一部分的值,或者传递回运行(调用)测试或组件的应用程序。

例如,假设要测试一个购物应用程序,该程序计算采购费用,并自动从账户中扣除采购金额。想要测试在每次运行带有不同的采购单的操作或组件时,该应用程序是否能够正确地从账户中扣除采购金额,可以将花费的总金额输出到某个操作或组件的参数值,然后在稍后的扣除该金额操作中的运行会话部分使用该值。

(2) 将值存储在运行时数据表中

对于要运行多次的由数据驱动的测试(或操作)来说,将值输出到运行时数据表的选项特别有用。在每次重复或循环中,QuickTest 检索当前值并将其存储在运行时数据表的相

应的行中。

例如，要测试一个航班预订应用程序，因此设计了一个测试来创建新预订，随后查看预订详细信息。每次运行测试时，应用程序为新预订生成一个唯一的订单号。要查看预订，应用程序要求用户输入相同的订单号。运行该测试之前，还不知道订单号。

要想解决这个问题，就要将在创建新预订时生成的唯一订单号的值输入数据表中。然后，在 View Reservation 屏幕中，使用包含存储值的列将输出值插入订单号输入字段中。运行测试时，QuickTest 检索站点为新预订生成的唯一订单号，并在运行时数据表中输入此输出值。测试到达查看预订所需的订单号输入字段时，QuickTest 将存储在运行时数据表中的唯一订单号插入订单号字段中。

(3) 将值存储在环境变量中

将值输出到内部用户定义的环境变量时，可以在运行会话后面的阶段使用该环境变量输入参数。

例如，假设在测试一个应用程序，该程序会提示用户在“欢迎使用”页输入账号，然后显示用户姓名，就可以使用文本输出值来捕获显示的名称值，并将其存储在环境变量中。最后，可以检索环境变量中的值以便在应用程序的其他位置中输入用户的姓名。

2. 输出属性值

1) 定义标准输出值

通过“输出值属性”对话框可以选择要输出的属性值，并定义选择的每个值的设置，如图 8-37 所示。

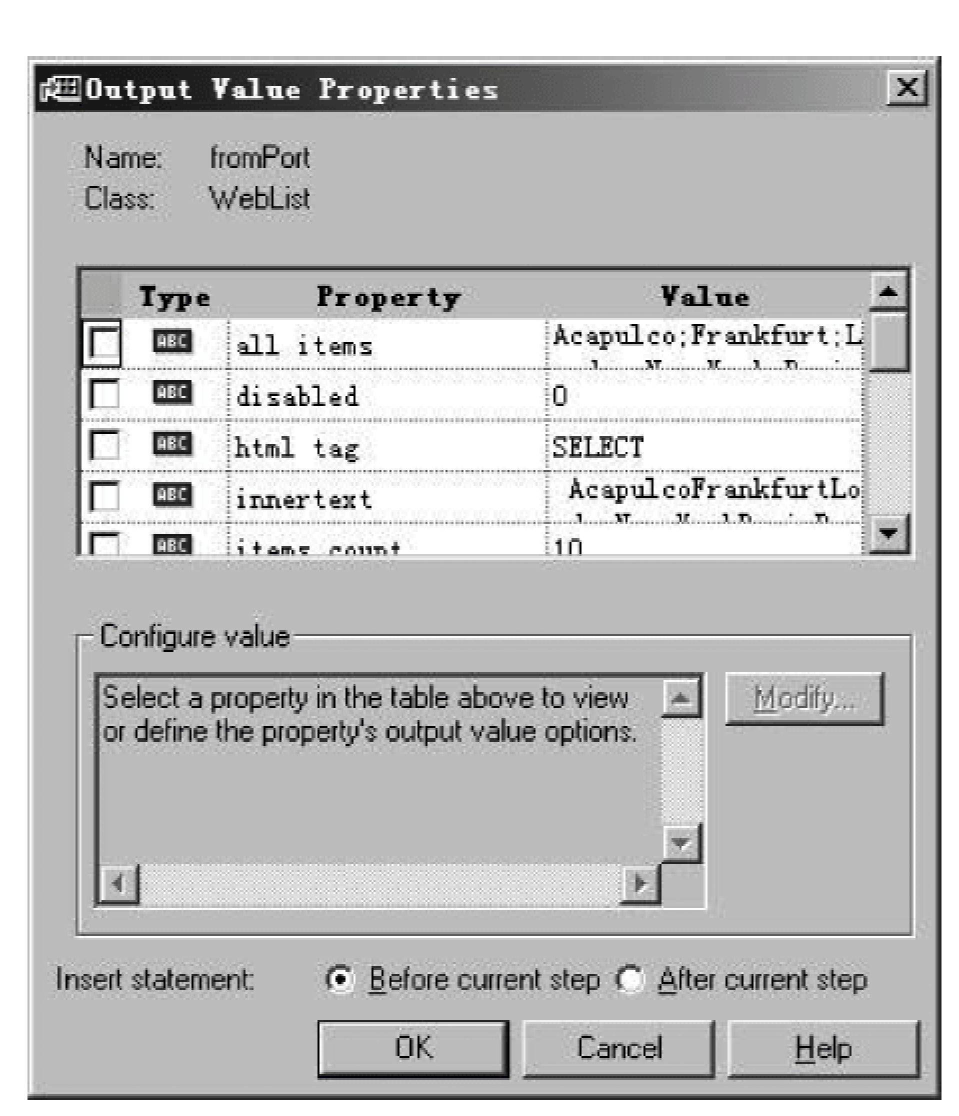


图 8-37 “输出值属性”对话框

关闭此对话框之前，可以为相同对象选择许多属性并为每个属性值定义输出设置。运行会话过程中到达输出值步骤时，QuickTest 将检索所有指定的属性值。

(1) 标识对象

对话框的上部显示有关要创建输出值的测试对象的信息，见表 8-6。

表 8-6　要创建输出值的测试对象的信息

项目	描　述
名称	测试对象的名称
类	对象的类别

（2）选择要输出的属性值

对话框的上半部分包含一个窗格，其中列出选定对象的属性，以及它们的值和类型。该窗格包含的项如表 8-7 所示。

表 8-7　窗格包含的项

窗口元素	描　述
复选框	要指定将输出的属性，选择相应的复选框，可以为对象选择多个属性，并为选择的每个属性值指定输出选项
类别	ABC 图标表示属性的值当前为常量 图标表示属性的值当前存储在测试、操作或组件参数中 图标表示属性的值当前存储在运行时数据表中 图标表示属性的值当前存储在环境变量中
属性	属性的名称
值	属性的当前值

（3）指定属性值的输出设置

选择属性的复选框时，将突出显示属性详细信息，并且在“配置值”区域中显示选定属性值的当前输出定义，如图 8-38 所示。

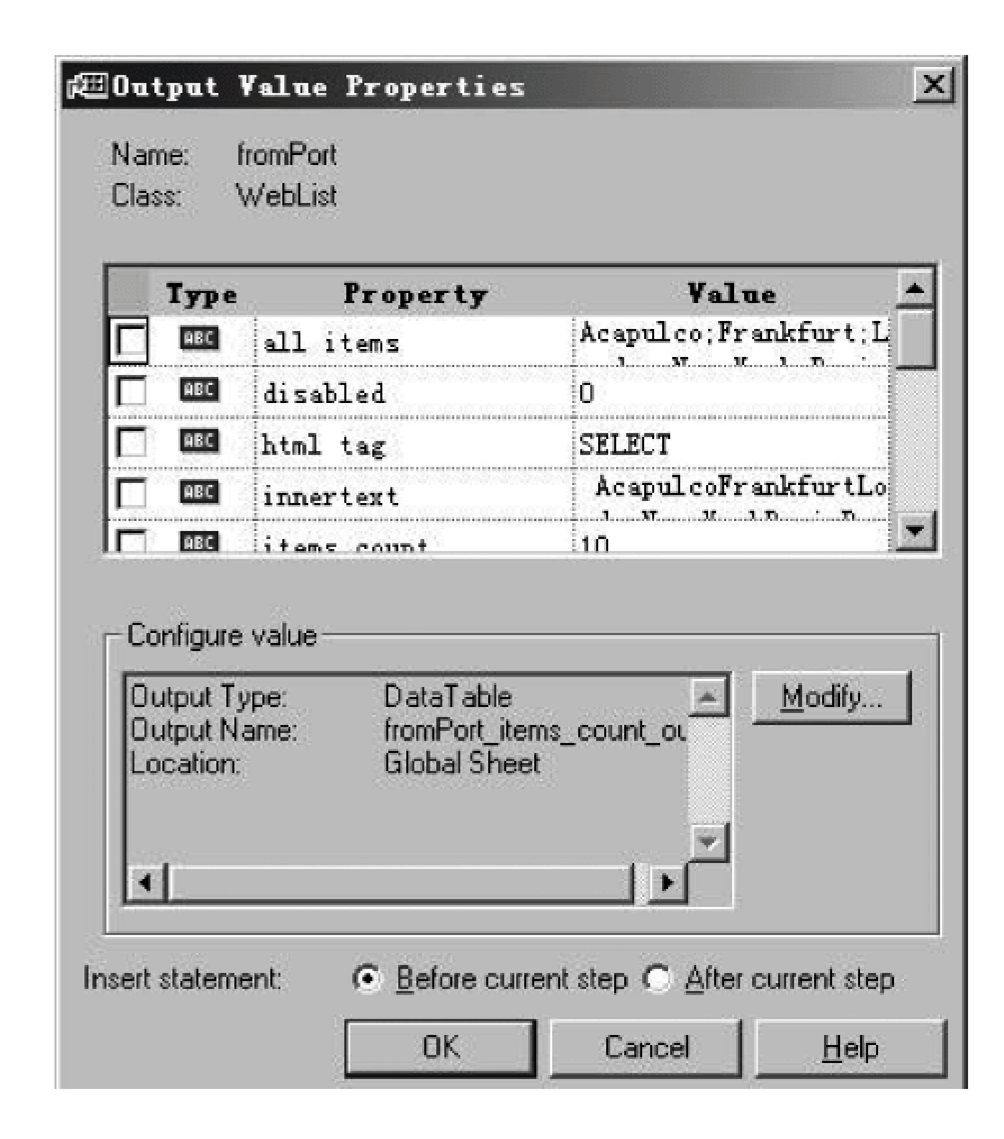


图 8-38　“配置值”区域

第一次选择要输出的属性值时，“配置值”区域中会显示值的默认输出定义。选择要输出的属性值时，可以：

① 通过选择其他属性值或单击 OK 按钮接受显示的输出定义。

② 通过单击“修改”按钮更改选定值的输出类型和/或设置。将打开“输出选项”对话框并显示该值当前的输出类型和设置。

2）指定输出类型和设置

为每个值定义的输出类型和设置决定该值在运行会话中的存储位置以及使用方式。到达输出值步骤时，QuickTest 检索为输出选定的每个值并将其存储在指定位置，以供以后在运行会话中使用。新建输出值步骤时，QuickTest 为选定要输出的每个值指定一个默认定义。

可以通过选择不同的输出类型并/或更改输出设置来更改选定值的当前输出定义。

(1) 将值输出到操作或组件参数

可以将值输出到操作或组件参数,以便这些值可以在运行会话的后面部分中使用,或者传递回运行(调用)测试或组件的外部应用程序。如果参数已经定义为用于调用操作或组件的输出参数,只能将值输出到操作或组件参数。此外,仅当输出值类型和参数值类型匹配时,将值输出到操作或组件的选项才可用。选择"测试参数"、"操作参数"或"组件参数"作为输出类型时,通过"输出选项"对话框可以选择在其中存储运行会话持续时间的选定值的参数。

(2) 将值输出到数据表

选择"数据表"作为输出类型时,通过"输出选项"对话框可以指定在运行时数据表中存储选定值的位置,如图 8-39 所示。

在将值输出到数据表时,有以下选项可以修改。

① 名称:指定数据表中要存储值的列的名称。QuickTest 建议使用输出的默认名称。可以从列表中选择现有的输出名称,也可以通过使用默认输出名称或输入有效的描述性名称来新建输出名称。

② 数据表中的位置:输出测试的值时,指定将数据表列名称添加到数据表的全局工作表还是当前操作工作表中。

(3) 将值输出到环境变量

如果选择"环境"作为输出类型时,通过"输出选项"对话框,如图 8-40 所示,可以指定要在其中存储运行会话持续时间的选定值的环境参数,该参数由内部用户定义。



图 8-39 "输出选项"对话框(1)



图 8-40 "输出选项"对话框(2)

3. 在脚本中建立输出值

1) 建立输出值

因为在表格检查点中机票价钱的预期结果,并没有随着出发地点的改变而变动,导致第二、第三次的执行结果是失败的。

现在,从 Select a Flight: Mercury 网页上取得机票价钱,并且以取得的机票价钱更新表格检查点的预期结果,这样一来,测试脚本就可以利用在 Select a Flight: Mercury 网页上取得的机票价钱去验证 Book a Flight: Mercury 上显示的机票价钱。

(1) 打开 Parameter 测试脚本,将脚本另存为 Output 测试脚本。

(2) 在树视图中,展开 Welcome: Mercury Tours 并且单击 Select a Flight: Mercury 网页,在 Active Screen 窗口中会显示相应的页面。在 Active Screen 窗口中选取 270,然后单击鼠标右键,选择 Insert Text Output 选项,打开 Text Output Value Properties 对话框,如图 8-41 所示。

(3) 在 Text Output Value Properties 对话框中单击 Modify 按钮,打开 Output Options 对话框,如图 8-42 所示。

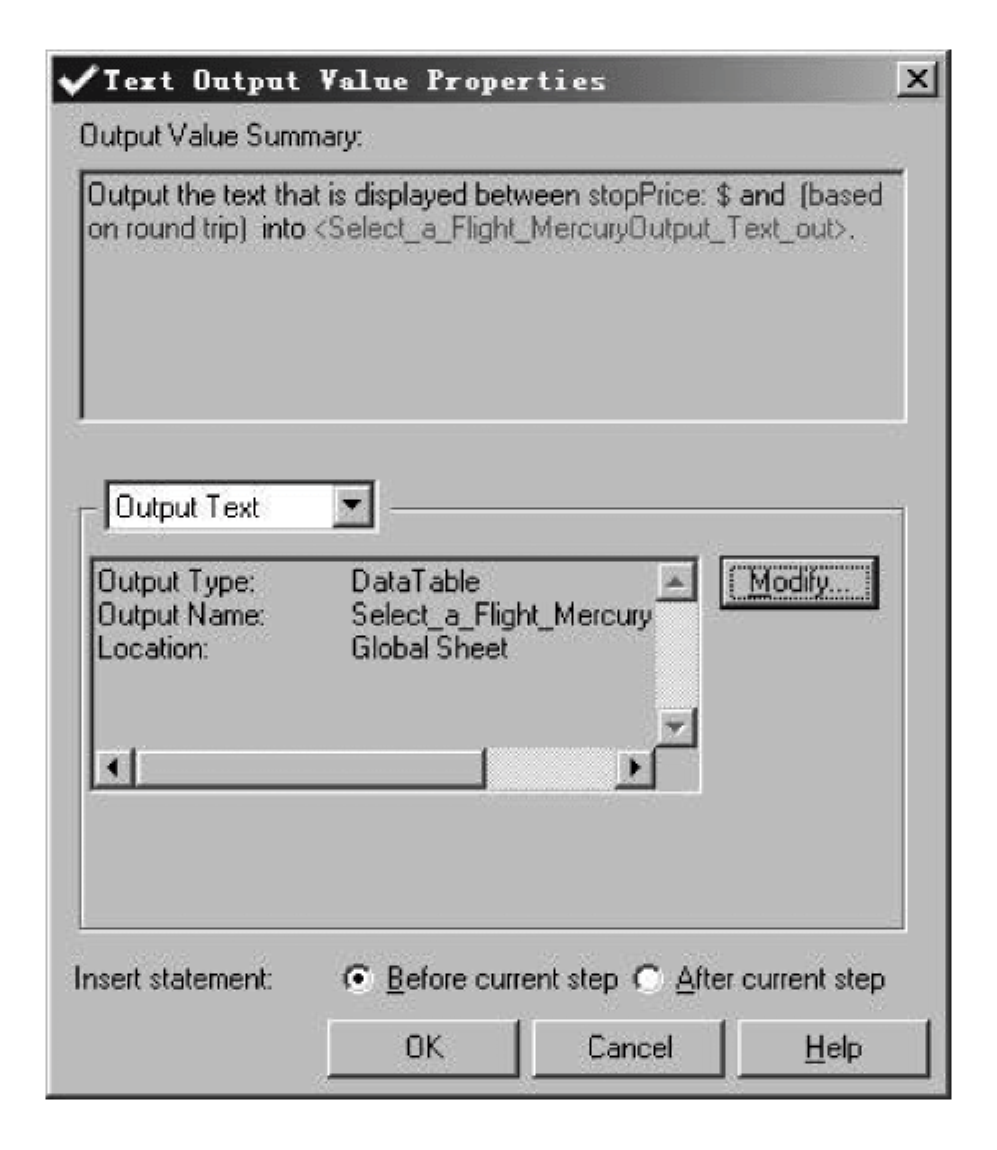


图 8-41 Text Output Value Properties 对话框

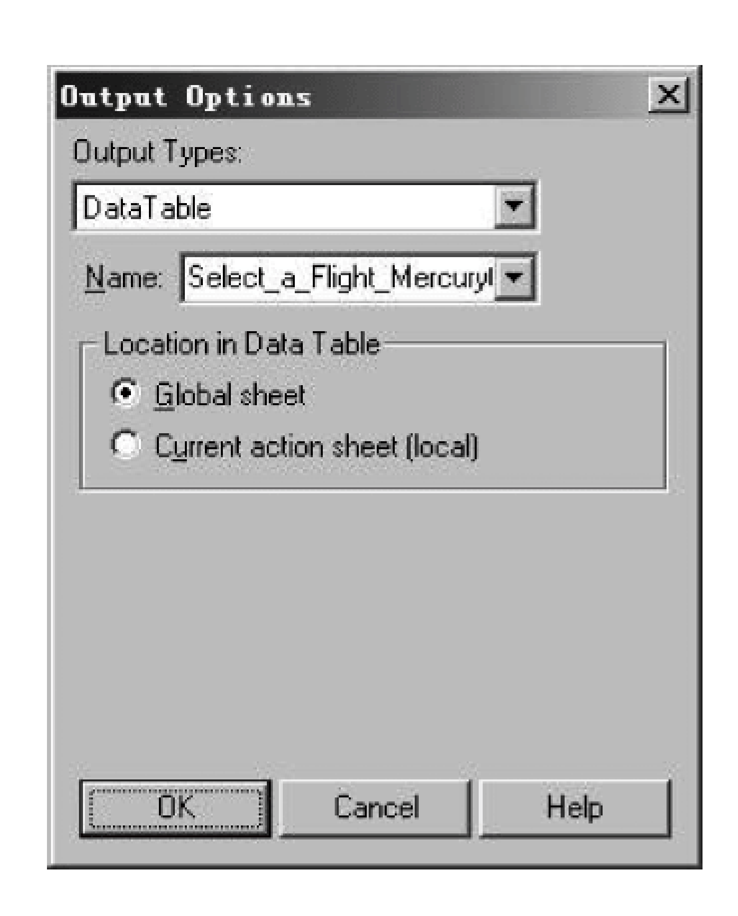


图 8-42 Output Options 对话框

在 Name 字段显示 Select_a_Flight_MercuryOutput_Text_out,将其改成"depart_flight_price",接受其他默认值,单击 OK 按钮确认,QuickTest 会在 DataTable 中加入 depart_flight_price 字段。

在 DataTable 上的 depart_flight_price 字段的第一行会显示从应用程序上取得的输出值(270)。在执行时,第一次 QuickTest 会取得一样的值 270,接下来的第二、第三次会从应用程序上取得实际值,并存放在 DataTable 中。

(4) 修正表格检查点的预期值。

在树视图中,展开 Welcome: Mercury Tours > Book a Flight: Mercury,在 Checkpoint"New York to San Francisco"上单击鼠标右键,选择 Checkpoint Properties 选项,打开 Table Checkpoint Properties 对话框。

选中第三行、第三列(被勾选的字段),在 Configure value 中选择 Parameter,然后单击 Parameter Options 按钮,打开 Parameter Options 对话框,如图 8-43 所示。

在"名字"下拉列表中选择 depart_flight_price。

(5) 单击 OK 按钮回到 Table Checkpoint Properties 对话框,如图 8-44 所示,可以看到这个检查点的预期结果已经被参数化了。

单击 OK 按钮关闭 Table Checkpoint Properties 对话框,保存测试脚本。

2）执行并分析使用输出值的测试脚本

前面在脚本中建立了输出值，并且将表格检查点参数化，现在执行 Output 测试脚本。

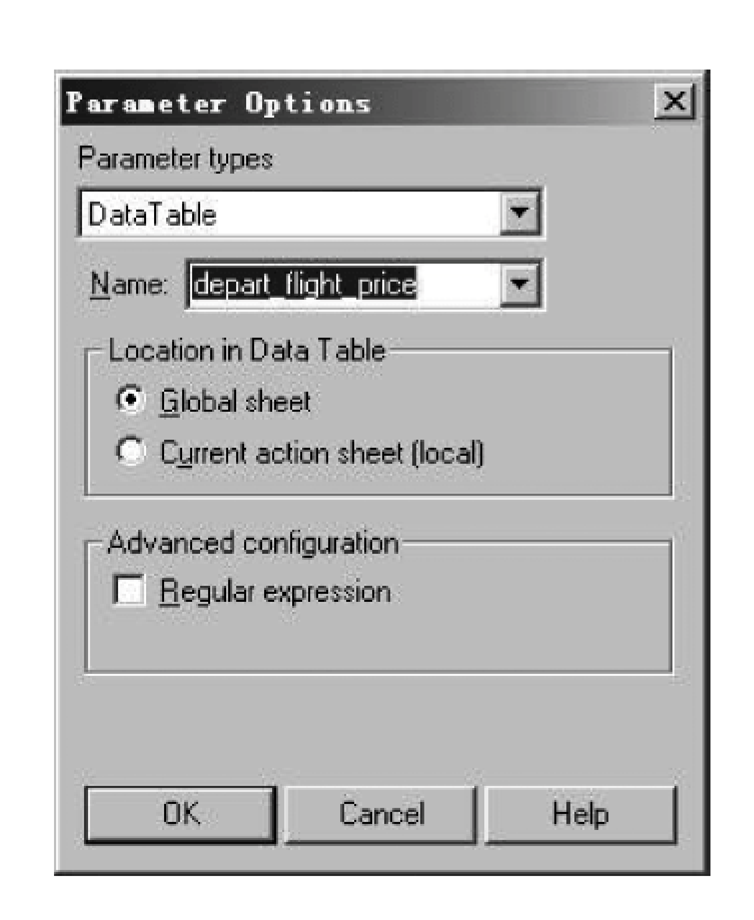


图 8-43 Parameter Options 对话框

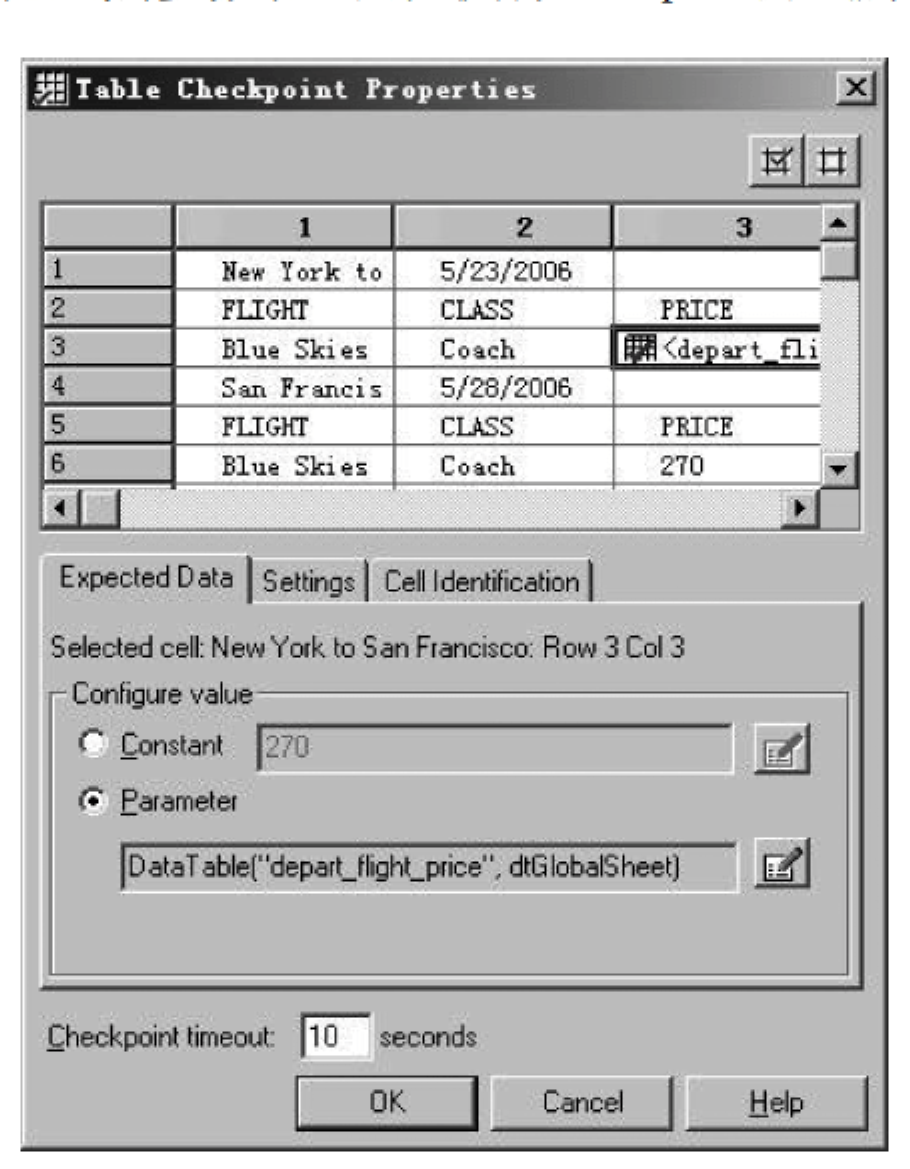


图 8-44 Table Checkpoint Properties 对话框

执行测试脚本：单击工具栏上的 Run 按钮，开启 Run 对话框，选取 New run results folder，其余为默认值，单击 OK 按钮开始执行脚本。当脚本运行结束后，会开启测试结果窗口。

在执行结果窗口中，单击树视图中的 Run-Time-Data，如图 8-45 所示，可以在表格中看到执行测试时使用的输出值，在 depart_flight_price 字段中显示了不同的机票价钱。

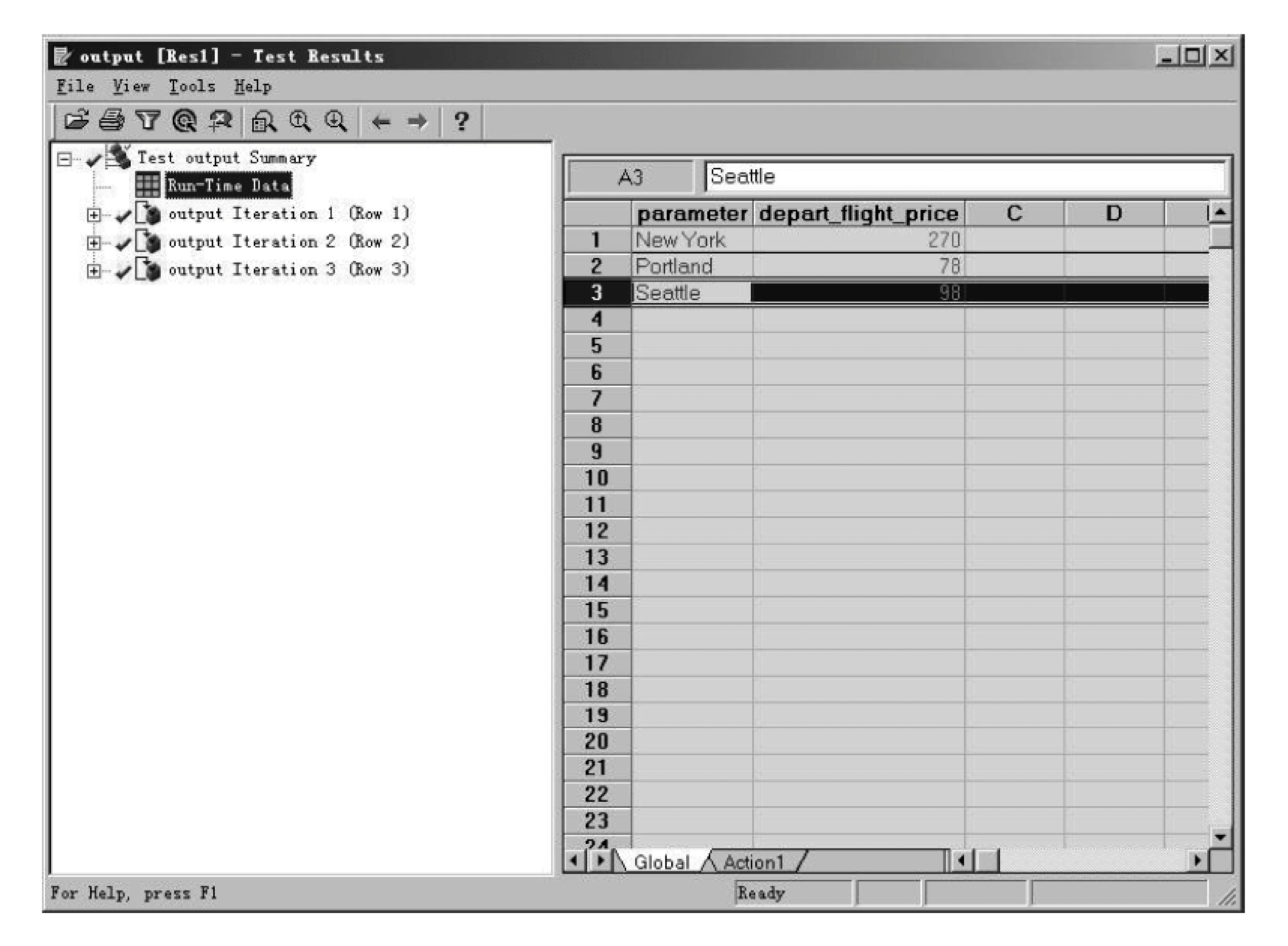


图 8-45 执行结果窗口

在结果窗口中单击 Test output Summary 可以看到,12 个检测点都通过了验证,运行结果均为 Passed。

8.2 白盒测试工具

白盒测试也称结构测试或逻辑驱动测试,它是按照程序内部的结构测试程序,通过测试来检测产品内部动作是否按照设计规格说明书的规定正常进行,检验程序中的每条通路是否都能按预定要求正确工作。这一方法是把测试对象看作一个打开的盒子,测试人员依据程序内部逻辑结构相关信息,设计或选择测试用例,对程序所有逻辑路径进行测试,通过在不同点检查程序的状态,确定实际的状态是否与预期的状态一致。本节以 JUnit 为例,讲述如何使用 JUint 对 Java 程序进行白盒测试。

8.2.1 JUnit 简介

JUnit 是一个 Java 语言的单元测试框架。它由 Kent Beck 和 Erich Gamma 建立,逐渐成为源于 Kent Beck 的 sUnit 的 xUnit 家族中最为成功的一个。JUnit 是在极限编程和重构(Refactor)中被极力推荐使用的工具,因为在实现自动单元测试的情况下可以大大提高开发的效率,但是实际上编写测试代码也是需要耗费很多的时间和精力的。

1. 对于极限编程而言

要求在编写代码之前先写测试,这样可以强制程序员在写代码之前好好地思考代码(方法)的功能和逻辑,否则编写的代码会很不稳定,那么就需要同时维护测试代码和实际代码,这个工作量就会大大增加。因此在极限编程中,基本过程是这样的：构思→编写测试代码→编写代码→测试,而且编写测试和编写代码都是增量式的,写一点儿测一点儿,在编写以后的代码中如果发现问题可以较快地追踪到问题的原因,减小回归错误的纠错难度。

2. 对于重构而言

其好处和极限编程中是类似的,因为重构也是要求改一点儿测一点儿,减少回归错误造成的时间消耗。

3. 其他情况

在开发的时候使用 JUnit 写一些适当的测试也是有必要的,因为一般也是需要编写测试的代码的,可能原来不是使用 JUnit,如果使用 JUnit,而且针对接口(方法)编写测试代码会减少以后的维护工作。例如,以后对方法内部的修改(这个就是相当于重构的工作了)。另外就是因为 JUnit 有断言功能,如果测试结果不通过会告诉我们哪个测试不通过以及为什么,而如果是像以前的一般做法写一些测试代码看其输出结果,然后再由自己来判断结果是否正确,使用 JUnit 的好处就是这个结果是否正确的判断是由它来完成的,我们只需要看看它告诉我们结果是否正确就可以了,在一般情况下会大大提高效率。

8.2.2 JUnit 的使用

在编写大型程序的时候，需要写成千上万个方法或函数，这些函数的功能可能很强大，但在程序中只用到该函数的一小部分功能，并且经过调试可以确定这一小部分功能是正确的。但是，同时应该确保每一个函数都完全正确，因为如果今后对程序进行扩展，用到了某个函数的其他功能，而这个功能有 Bug，那绝对是一件非常郁闷的事情。所以说，每编写完一个函数之后，都应该对这个函数的方方面面进行测试，这样的测试称为单元测试。传统的编程方式进行单元测试是一件很麻烦的事情，要重新写另外一个程序，在该程序中调用需要测试的方法，并且仔细观察运行结果，看看是否有错。正因为如此麻烦，所以程序员们编写单元测试的热情不是很高。于是有一个牛人推出了单元测试包，大大简化了进行单元测试所要做的工作，这就是 JUnit4。本节简要介绍一下在 Eclipse 3.2 中使用 JUnit4 进行单元测试的方法。

首先，我们来一个傻瓜式速成教程，不要问为什么，先来体验一下单元测试的快感！

第一步，新建一个项目叫 JUnit_Test，编写一个 Calculator 类，这是一个能够简单实现加减乘除、平方、开方的计算器类，然后对这些功能进行单元测试。这个类并不是很完美，我们故意保留了一些 Bug 用于演示，这些 Bug 在注释中都有说明。该类代码如下。

```
package andycpp;
public class Calculator ...{
    private static int result;                  // 静态变量,用于存储运行结果
public void add(int n) ...{
result = result + n;
    }
public void substract(int n) ...{
        result = result - 1;                    //Bug: 正确的应该是 result = result - n
    }
public void multiply(int n) ...{
    }                                           // 此方法尚未写好
public void divide(int n) ...{
result = result / n;
    }
public void square(int n) ...{
result = n * n;
    }
public void squareRoot(int n) ...{
        for (; ;) ;                             //Bug : 死循环
    }
public void clear() ...{                        // 将结果清零
result = 0;
    }
public int getResult() ...{
return result;
    }
}
```

第二步，将 JUnit4 单元测试包引入这个项目：在该项目上单击右键，选择“属性”选项，如图 8-46 所示。

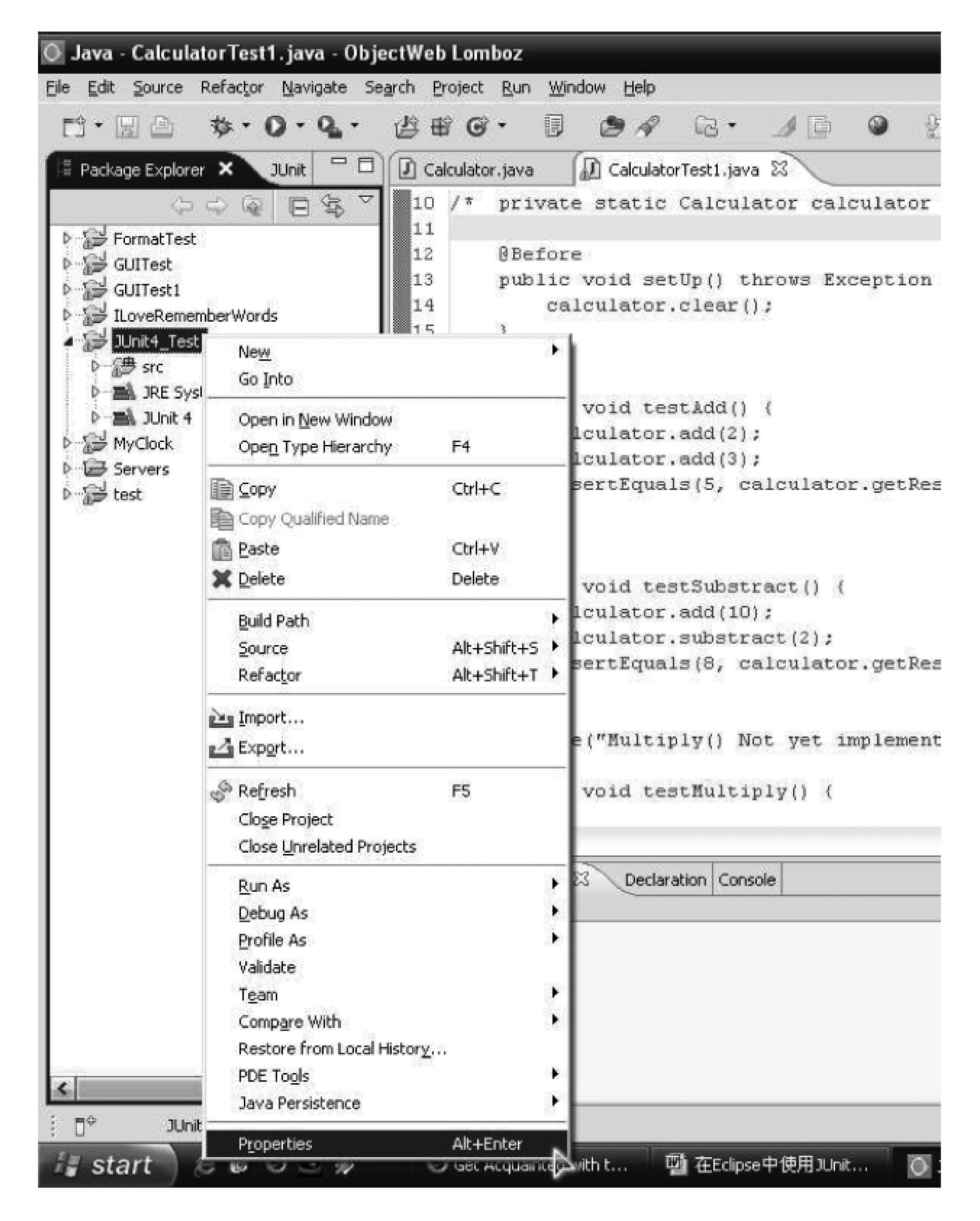


图 8-46　将 JUnit4 单元测试包引入项目

在弹出的属性窗口中，首先在左边选择 Java Build Path，然后到右上选择 Libraries 选项卡，之后在最右边单击 Add Library…按钮，如图 8-47 所示。

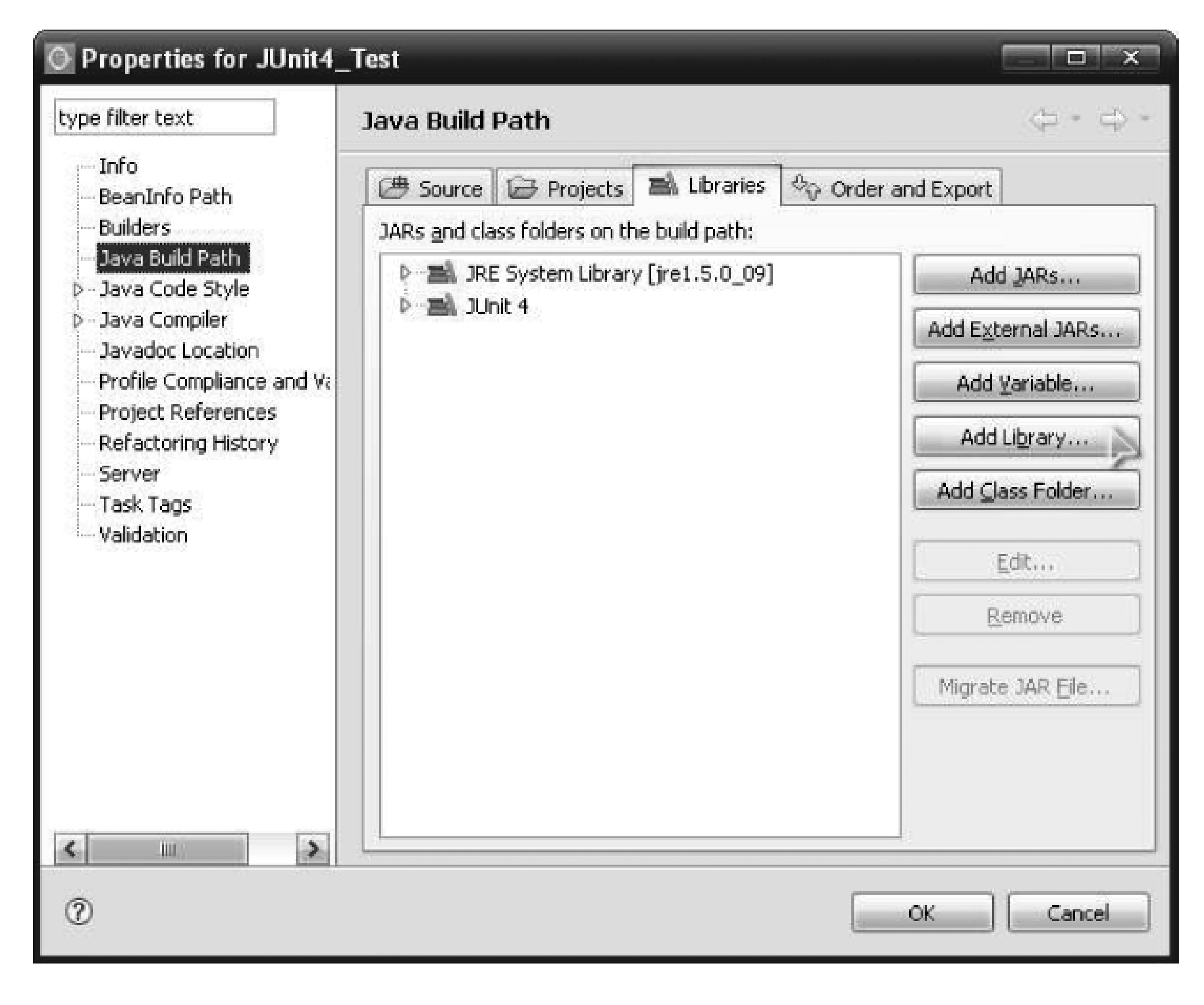


图 8-47　“属性”窗口

然后在新弹出的对话框中选择 JUnit4 并单击“确定”按钮，如图 8-47 所示，JUnit4 软件包就被包含进这个项目了。

第三步，生成 JUnit 测试框架：在 Eclipse 的 Package Explorer 中用右键单击该类弹出菜单，选择 New→JUnit Test Case，如图 8-48 所示。

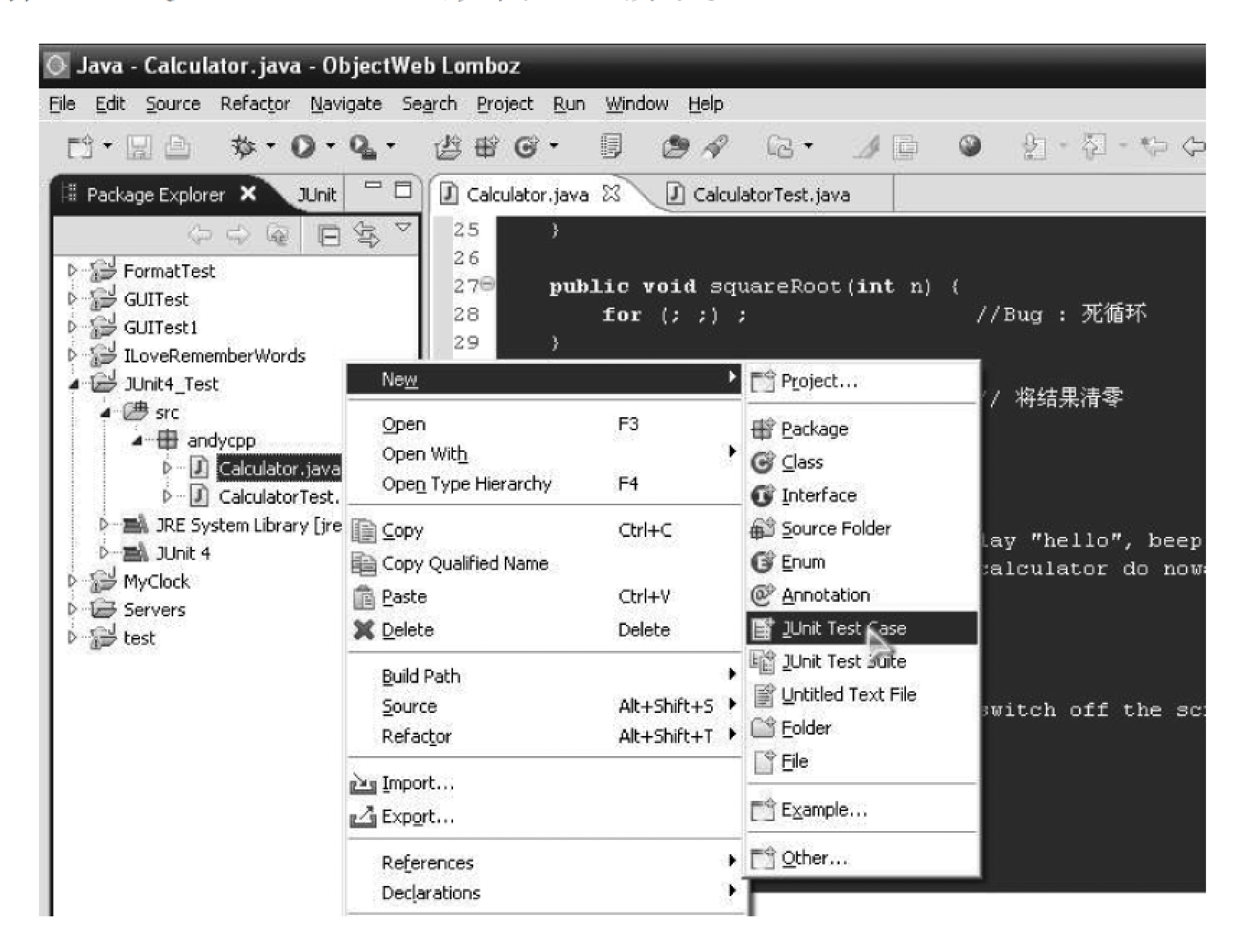


图 8-48 生成 JUnit 测试框架

在弹出的对话框中，进行相应的选择，如图 8-49 所示。



图 8-49 在对话框中进行选择

单击 Next 按钮后，系统会自动列出这个类中包含的方法，选择要进行测试的方法。此例中仅对加、减、乘、除 4 个方法进行测试，如图 8-50 所示。

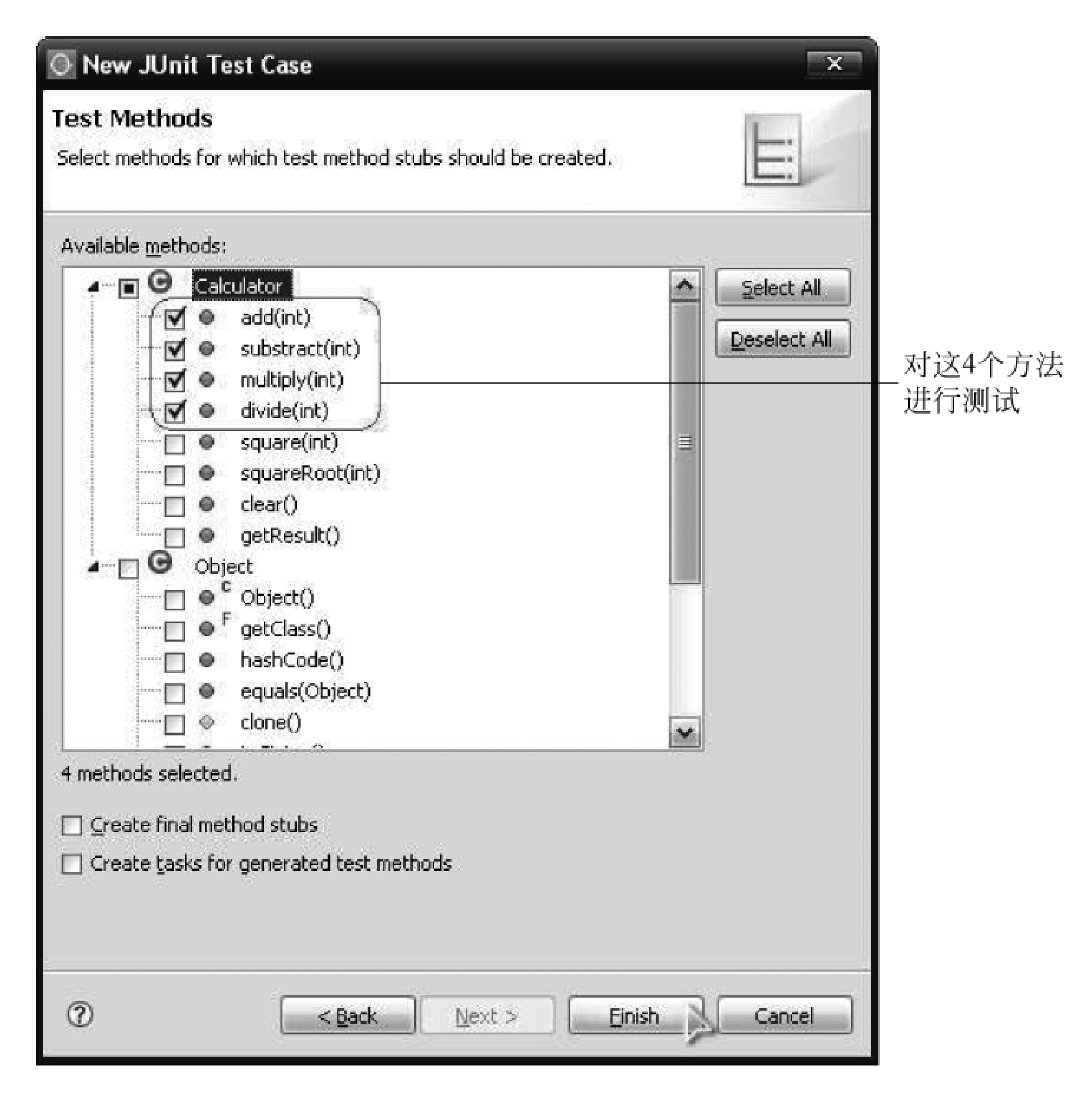


图 8-50 对加、减、乘、除 4 个方法进行测试

之后系统会自动生成一个新类 CalculatorTest，里面包含一些空的测试用例。只需要将这些测试用例稍做修改即可使用。完整的 CalculatorTest 代码如下。

```
package andycpp;
import static org.junit.Assert.*;
import org.junit.Before;
import org.junit.Ignore;
import org.junit.Test;
public class CalculatorTest ...{
private static Calculator calculator = new Calculator();
    @Before
public void setUp() throws Exception ...{
calculator.clear();
    }
    @Test
public void testAdd() ...{
calculator.add(2);
calculator.add(3);
assertEquals(5, calculator.getResult());
    }
    @Test
public void testSubstract() ...{
calculator.add(10);
```

```
calculator.substract(2);
assertEquals(8, calculator.getResult());
    }
@Ignore("Multiply() Not yet implemented")
    @Test
public void testMultiply() ...{
    }
    @Test
public void testDivide() ...{
calculator.add(8);
calculator.divide(2);
assertEquals(4, calculator.getResult());
    }
}
```

第四步，运行测试代码：按照上述代码修改完毕后，在 CalculatorTest 类上单击右键，选择 Run As→JUnit Test 来运行测试，如图 8-51 所示。

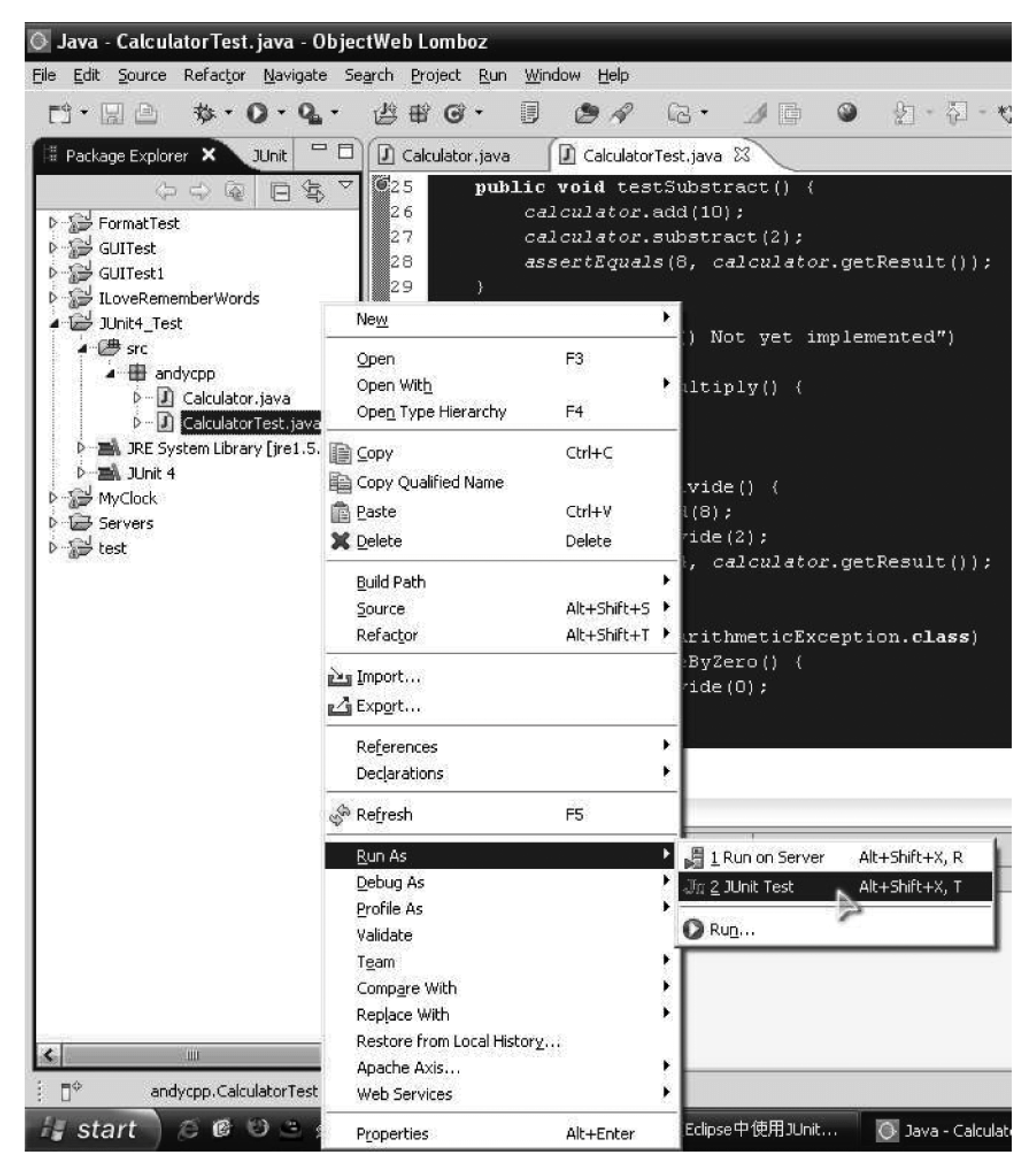


图 8-51 运行测试

运行结果如图 8-52 所示。

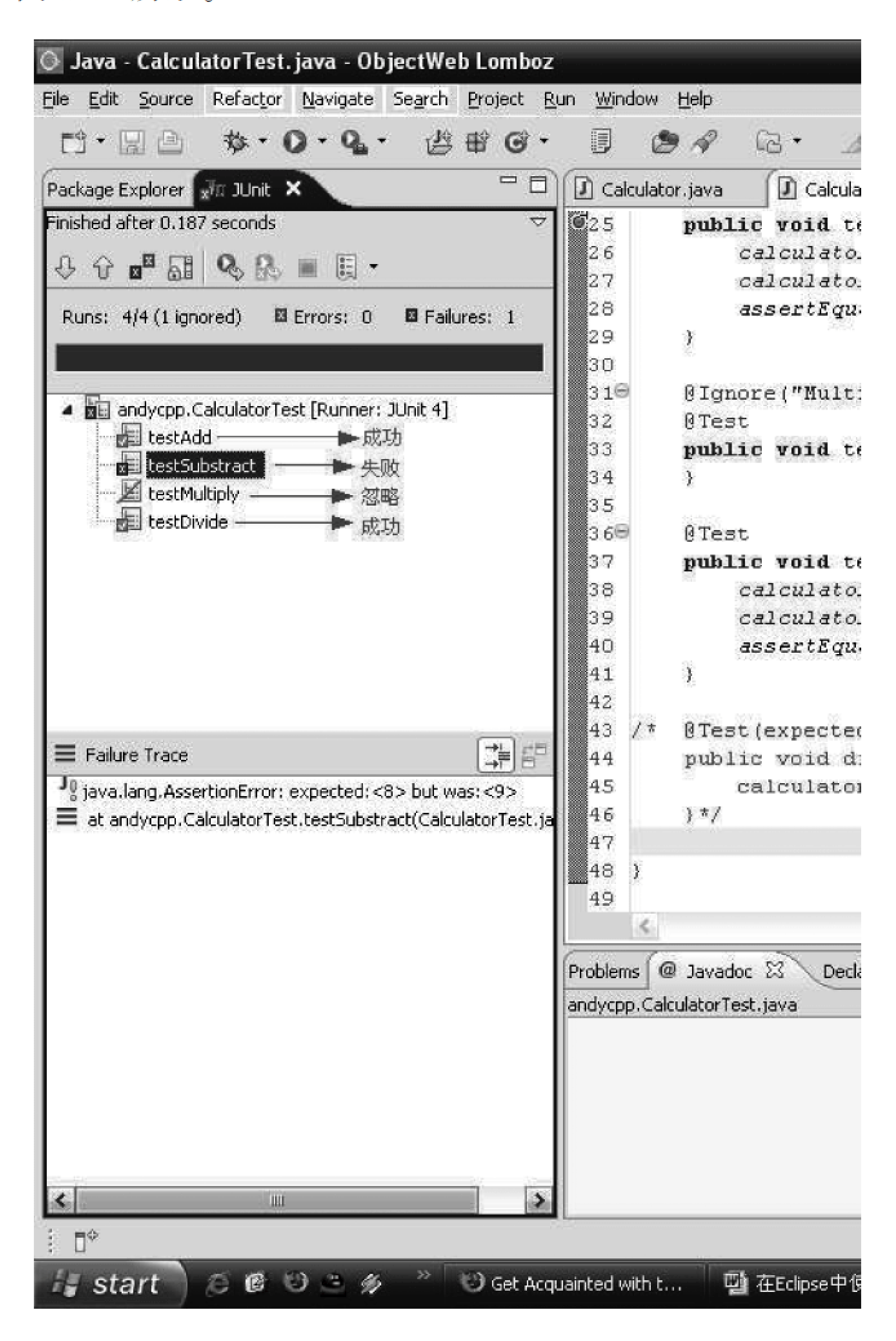


图 8-52　运行结果

进度条是红颜色表示发现错误，具体的测试结果在进度条上面有表示“共进行了 4 个测试，其中一个测试被忽略，一个测试失败”。

至此，已经完整体验了在 Eclipse 中使用 JUnit 的方法。在运用中，使用以下几个步骤。

1. 包含必要的 Package

在测试类中用到了 JUnit4 框架，自然要把相应的 Package 包含进来。最主要的一个 Package 就是 org.junit.*。把它包含进来之后，绝大部分功能就有了。还有一句话也非常重要“import static org.junit.Assert.*;”，在测试的时候使用的一系列 assertEquals 方法就来自这个包。注意，这是一个静态包含(static)，是 JDK5 中新增添的一个功能。也就是说，assertEquals 是 Assert 类中的一系列的静态方法，一般的使用方式是 Assert.assertEquals()，但是使用了静态包含后，前面的类名就可以省略了，使用起来更加方便。

2. 测试类的声明

注意,我们的测试类是一个独立的类,没有任何父类。测试类的名字也可以任意命名,没有任何局限性。所以不能通过类的声明来判断它是不是一个测试类,它与普通类的区别在于它内部的方法的声明,下面会讲到。

3. 创建一个待测试的对象

要测试哪个类,那么首先就要创建一个该类的对象。正如前文中的代码:

```
private static Calculator calculator = new Calculator();
```

为了测试 Calculator 类,必须创建一个 calculator 对象。

4. 测试方法的声明

在测试类中,并不是每一个方法都是用于测试的,必须使用“标注”来明确表明哪些是测试方法。“标注”也是 JDK5 的一个新特性,用在此处非常恰当。可以看到,在某些方法的前面有@Before、@Test、@Ignore 等字样,这些就是标注,以一个“@”作为开头。这些标注都是 JUnit4 自定义的,熟练掌握这些标注的含义非常重要。

5. 编写一个简单的测试方法

首先,要在方法的前面使用@Test 标注,以表明这是一个测试方法。对于方法的声明也有如下要求:名字可以随便取,没有任何限制,但是返回值必须为 void,而且不能有任何参数。如果违反这些规定,会在运行时抛出一个异常。至于方法内该写些什么,那就要看需要测试些什么了。例如:

```
    @Test
public void testAdd() …{
calculator.add(2);
calculator.add(3);
assertEquals(5, calculator.getResult());
    }
```

如果想测试一下“加法”功能是否正确,就在测试方法中调用几次 add 函数,初始值为 0,先加 2,再加 3,期待的结果应该是 5。如果最终实际结果也是 5,则说明 add 方法是正确的,反之说明它是错的。assertEquals(5, calculator.getResult());就是来判断期待结果和实际结果是否相等,第一个参数填写期待结果,第二个参数填写实际结果,也就是通过计算得到的结果。这样写好之后,JUnit 会自动进行测试并把测试结果反馈给用户。

6. 忽略测试某些尚未完成的方法

如果在写程序前做了很好的规划,那么哪些方法是什么功能都应该定下来。因此,即使该方法尚未完成,它的具体功能也是确定的,这也就意味着可以为它编写测试用例。但是,如果已经把该方法的测试用例写完,但该方法尚未完成,那么测试的时候一定是“失败”。这种失败和真正的失败是有区别的,因此 JUnit 提供了一种方法来区别它们,那就是在这种测

试函数的前面加上@Ignore标注，这个标注的含义就是“某些方法尚未完成，暂不参与此次测试”。这样测试结果就会提示有几个测试被忽略，而不是失败。一旦完成了相应函数，只需要把@Ignore标注删除，就可以进行正常的测试。

7. Fixture(固定代码段)

Fixture的含义就是“在某些阶段必然被调用的代码”。例如上面的测试，由于只声明了一个Calculator对象，它的初始值是0，但是测试完加法操作后，它的值就不是0了；接下来测试减法操作，就必然要考虑上次加法操作的结果。如果这样设计就糟了！我们非常希望每一个测试都是独立的，相互之间没有任何耦合度。因此，就很有必要在执行每一个测试之前，对Calculator对象进行一个“复原”操作，以消除其他测试造成的影响。因此，“在任何一个测试执行之前必须执行的代码”就是一个Fixture，用@Before来标注它，如前面例子所示：

```
    @Before
public void setUp() throws Exception ...{
calculator.clear();
    }
```

这里不再需要@Test标注，因为这不是一个Test，而是一个Fixture。同理，如果“在任何测试执行之后需要进行的收尾工作”也是一个Fixture，使用@After来标注。由于本例比较简单，没有用到此功能。

上文介绍了两个Fixture标注，分别是@Before和@After，下面来看看它们是否适合完成如下功能：有一个类是负责对大文件(超过500MB)进行读写，它的每一个方法都是对文件进行操作。换句话说，在调用每一个方法之前，都要打开一个大文件并读入文件内容，这绝对是一个非常耗费时间的操作。如果使用@Before和@After，那么每次测试都要读取一次文件，效率极其低下。这里所希望的是在所有测试一开始读一次文件，所有测试结束之后释放文件，而不是每次测试都读文件。JUnit的作者显然也考虑到了这个问题，它给出了@BeforeClass和@AfterClass两个Fixture来实现这个功能。从名字上就可以看出，用这两个Fixture标注的函数，只在测试用例初始化时执行@BeforeClass方法，当所有测试执行完毕之后，执行@AfterClass进行收尾工作。在这里要注意一下，每个测试类只能有一个方法被标注为@BeforeClass或@AfterClass，并且该方法必须是Public和Static的。

8. 限时测试

例如，求平方根的函数有Bug，是个死循环：

```
public void squareRoot(int n) ...{
        for (; ;) ;                          //Bug : 死循环
    }
```

如果测试的时候遇到死循环，一般会导致程序无法自行中止。因此，对于那些逻辑很复杂，循环嵌套比较深的程序，很有可能出现死循环，因此一定要采取一些预防措施。限时测试是一个很好的解决方案，给这些测试函数设定一个执行时间，超过了这个时间，它们就会被系统强行终止，并且系统还会汇报该函数结束的原因是因为超时，这样就可以发现这些

Bug 了。要实现这一功能,只需要给@Test 标注加一个参数即可,代码如下。

```
@Test(timeout = 1000)
public void squareRoot() ...{
calculator.squareRoot(4);
assertEquals(2, calculator.getResult());
}
```

Timeout 参数表明了要设定的时间,单位为 ms,因此 1000 就代表 1s。

9. 测试异常

Java 中的异常处理也是一个重点,因此经常会编写一些需要抛出异常的函数。那么,如果觉得一个函数应该抛出异常,但是它没抛出,这算不算 Bug 呢? 这当然是 Bug。JUnit 也考虑到了这一点,来帮助我们找到这种 Bug。例如,计算器类有除法功能,如果除数是一个 0,那么必然要抛出"除 0 异常"。因此,很有必要对这些进行测试,代码如下。

```
@Test(expected = ArithmeticException.class)
public void divideByZero() ...{
calculator.divide(0);
   }
```

如上述代码所示,需要使用@Test 标注的 expected 属性,将要检验的异常传递给它,这样 JUnit 框架就能自动检测是否抛出了指定的异常。

10. Runner(运行器)

读者有没有想过这个问题,当把测试代码提交给 JUnit 框架后,框架如何来运行代码呢? 答案就是——Runner。在 JUnit 中有很多个 Runner,它们负责调用测试代码,每一个 Runner 都有各自的特殊功能,要根据需要选择不同的 Runner 来运行测试代码。可能读者会觉得奇怪,前面写了那么多测试,并没有明确指定一个 Runner 啊? 这是因为 JUnit 中有一个默认 Runner,如果没有指定,那么系统自动使用默认 Runner 来运行代码。换句话说,下面两段代码含义是完全一样的。

```
import org.junit.internal.runners.TestClassRunner;
import org.junit.runner.RunWith;
//使用了系统默认的 TestClassRunner,与下面代码完全一样
public class CalculatorTest ...{...}
@RunWith(TestClassRunner.class)
public class CalculatorTest ...{...}
```

从上述例子可以看出,要想指定一个 Runner,需要使用@RunWith 标注,并且把所指定的 Runner 作为参数传递给它。另外一个要注意的是,@RunWith 是用来修饰类的,而不是用来修饰函数的。只要对一个类指定了 Runner,那么这个类中的所有函数都被这个 Runner 来调用。最后,不要忘了包含相应的 Package,上面的例子对这一点写得很清楚了。接下来,展示其他 Runner 的特有功能。

11. 参数化测试

可能遇到过这样的函数,它的参数有许多特殊值,或者说它的参数分为很多个区域。例

如，一个对考试分数进行评价的函数，返回值分别为“优秀、良好、一般、及格、不及格”，因此在编写测试的时候，至少要写 5 个测试，把这 5 种情况都包含进去，这确实是一件很麻烦的事情。还使用先前的例子，测试一下“计算一个数的平方”这个函数，暂且分为三类：正数、0、负数。测试代码如下。

```
import org.junit.AfterClass;
import org.junit.Before;
import org.junit.BeforeClass;
import org.junit.Test;
import static org.junit.Assert.*;
public class AdvancedTest ...{
private static Calculator calculator = new Calculator();
    @Before
public void clearCalculator() ...{
calculator.clear();
}
    @Test
public void square1() ...{
calculator.square(2);
assertEquals(4, calculator.getResult());
}
@Test
public void square2() ...{
calculator.square(0);
assertEquals(0, calculator.getResult());
}
@Test
public void square3() ...{
calculator.square(-3);
assertEquals(9, calculator.getResult());
}
 }
```

为了简化类似的测试，JUnit4 提出了“参数化测试”的概念，只写一个测试函数，把这若干种情况作为参数传递进去，一次性地完成测试。代码如下。

```
import static org.junit.Assert.assertEquals;
import org.junit.Test;
import org.junit.runner.RunWith;
import org.junit.runners.Parameterized;
import org.junit.runners.Parameterized.Parameters;
import java.util.Arrays;
import java.util.Collection;
@RunWith(Parameterized.class)
public class SquareTest ...{
private static Calculator calculator = new Calculator();
private int param;
private int result;
@Parameters
public static Collection data() ...{
```

```
return Arrays.asList(new Object[][]...{
...{2, 4},
...{0, 0},
...{ - 3, 9},
        });
}
//构造函数,对变量进行初始化
public SquareTest(int param, int result) ...{
        this.param = param;
        this.result = result;
}
@Test
public void square() ...{
calculator.square(param);
assertEquals(result, calculator.getResult());
    }
 }
```

下面对上述代码进行分析。首先,要为这种测试专门生成一个新的类,而不能与其他测试共用同一个类,此例中定义了一个 SquareTest 类。然后,要为这个类指定一个 Runner,而不能使用默认的 Runner,因为特殊的功能要用特殊的 Runner。@RunWith(Parameterized.class)这条语句就是为这个类指定了一个 ParameterizedRunner。其次,定义一个待测试的类,并且定义两个变量,一个用于存放参数,一个用于存放期待的结果。再次,定义测试数据的集合,也就是上述的 data()方法,该方法可以任意命名,但是必须使用@Parameters 标注进行修饰。这个方法的框架就不予解释了,只需要注意其中的数据是一个二维数组,数据两两一组,每组中的这两个数据,一个是参数,一个是预期的结果。例如第一组{2, 4},2 就是参数,4 就是预期的结果。这两个数据的顺序无所谓,谁前谁后都可以。之后是构造函数,其功能就是对先前定义的两个参数进行初始化。在这里要注意一下参数的顺序,要和上面的数据集合的顺序保持一致。如果前面的顺序是{参数,期待的结果},那么构造函数的顺序也要是“构造函数(参数,期待的结果)”,反之亦然。最后就是写一个简单的测试用例,和前面介绍过的写法一样。

12. 打包测试

通过前面的介绍可以感觉到,在一个项目中,只写一个测试类是不可能的,我们会写出很多测试类。可是这些测试类必须一个一个地执行,这也是比较麻烦的事情。鉴于此,JUnit 提供了打包测试的功能,将所有需要运行的测试类集中起来,一次性运行完毕,大大方便了测试工作。具体代码如下。

```
import org.junit.runner.RunWith;
import org.junit.runners.Suite;
@RunWith(Suite.class)
@Suite.SuiteClasses(...{
        CalculatorTest.class,
        SquareTest.class
        })
public class AllCalculatorTests ...{}
```

可以看到，这个功能也需要使用一个特殊的 Runner，因此需要向@RunWith 标注传递一个参数 Suite. class。同时，还需要另外一个标注@Suite. SuiteClasses 来表明这个类是一个打包测试类。把需要打包的类作为参数传递给该标注就可以了。有了这两个标注之后，就已经完整地表达了所有的含义，因此下面的类已经无关紧要，随便起一个类名，内容全部为空即可。

思考题

1. 使用黑盒测试工具进行软件测试的侧重点是什么？
2. 使用 QTP 进行功能测试的测试流程是什么？
3. 举例说明如何使用 QTP 进行自动化测试。
4. 使用白盒测试工具进行软件测试的侧重点是什么？
5. 在 Eclipse 中使用 JUnit 需要经历哪几个步骤？
6. 举例说明在 Eclipse3.2 中使用 JUnit4 进行单元测试的方法。

第9章 性能测试工具与安全测试工具

本章学习重点

- 熟悉常见的性能测试工具软件 LoadRunner 的安装与应用。
- 熟悉常见的安全测试工具软件 AppScan 的安装与应用。

本章学习难点

- 熟悉常见的性能测试工具软件 LoadRunner 的应用。
- 熟悉常见的安全测试工具软件 AppScan 的应用。

9.1 性能测试工具

9.1.1 LoadRunner 简介

LoadRunner 是一种预测系统行为和性能的工业标准级负载测试工具。通过以模拟上千万用户实施并发负载及实时性能监测的方式来确认和查找问题，LoadRunner 能够对整个企业架构进行测试。通过使用 LoadRunner，企业能最大限度地缩短测试时间，优化性能和加速应用系统的发布周期。

目前企业的网络应用环境都必须支持大量用户，网络体系架构中包含各类应用环境且由不同供应商提供软件和硬件产品。难以预知的用户负载和愈来愈复杂的应用环境使公司时时担心会发生用户响应速度过慢、系统崩溃等问题。这些都不可避免地导致公司收益的损失。

Mercury Interactive 的 LoadRunner 能让企业保护自己的收入来源，最大限度地利用现有的 IT 资源而无须购置额外硬件，并确保终端用户在应用系统的各个环节中对其测试应用的质量、可靠性和可扩展性都有良好的评价。

LoadRunner 是一种适用于各种体系架构的自动负载测试工具，它能预测系统行为并优化系统性能。LoadRunner 的测试对象是整个企业的系统，它通过模拟实际用户的操作行为和实行实时性能监测，来帮助用户更快地查找和发现问题。此外，LoadRunner 能支持

广泛的协议和技术，为用户的特殊环境提供特殊的解决方案。

使用 LoadRunner 的 Virtual User Generator，能很简便地建立起系统负载。该引擎能够生成虚拟用户，以虚拟用户的方式模拟真实用户的业务操作行为。它先记录业务流程（如下订单或机票预订），然后将其转化为测试脚本。利用虚拟用户，可以在 Windows 、UNIX 或 Linux 机器上同时产生成千上万个用户访问。所以 LoadRunner 能极大地减少负载测试所需的硬件和人力资源。另外，LoadRunner 的 TurboLoad 专利技术能提供很高的适应性。TurboLoad 使用户可以承受每天几十万名在线用户和数以百万计的点击量的负载。

用 Virtual User Generator 建立测试脚本后，可以对其进行参数化操作，这一操作能让用户利用几套不同的实际发生数据来测试应用程序，从而反映出本系统的负载能力。以一个订单输入过程为例，参数化操作可将记录中的固定数据，如订单号和客户名称，由可变值来代替。在这些变量内随意输入可能的订单号和客户名来匹配多个实际用户的操作行为。

LoadRunner 通过它的 Data Wizard 来自动实现其测试数据的参数化。Data Wizard 直接连接数据库服务器，从中可以获取所需的数据（如订单号和用户名）并直接将其输入到测试脚本。这样避免了人工处理数据的需要，节省了大量的时间。

为了进一步确定虚拟用户能够模拟真实用户，可利用 LoadRunner 控制某些行为特性。例如，只需要单击鼠标，就能轻易控制交易的数量、交易频率、用户的思考时间和连接速度等。

虚拟用户建立后，需要设定负载方案、业务流程组合和虚拟用户数量。用 LoadRunner 的 Controller，能很快组织起多用户的测试方案。Controller 的 Rendezvous 功能提供一个互动的环境，在其中既能建立起持续且循环的负载，又能管理和驱动负载测试方案。而且，可以利用它的日程计划服务来定义用户在什么时候访问系统以产生负载。这样，就能将测试过程自动化。同样还可以用 Controller 来限定负载方案，在这个方案中所有的用户同时执行一个动作——如登录到一个库存应用程序——来模拟峰值负载的情况。另外，还能监测系统架构中各个组件的性能——包括服务器、数据库、网络设备等——来帮助客户决定系统的配置。

LoadRunner 通过它的 AutoLoad 技术，为用户提供更多的测试灵活性。使用 AutoLoad，可以根据目前的用户事先设定测试目标，优化测试流程。例如，用户的目标可以是确定应用系统承受的每秒点击量或每秒的交易量。

LoadRunner 内含集成的实时监测器，在负载测试过程的任何时候，用户都可以观察到应用系统的运行性能。这些性能监测器实时显示交易性能数据（如响应时间）和其他系统组件，包括 Application Server、Web Server、网络设备和数据库等的实时性能。这样，就可以在测试过程中从客户和服务器的双方面评估这些系统组件的运行性能，从而更快地发现问题。

再者，利用 LoadRunner 的 ContentCheck TM，可以判断负载下的应用程序功能正常与否。ContentCheck 在虚拟用户运行时，检测应用程序的网络数据包内容，从中确定是否有错误内容传送出去。它的实时浏览器帮助用户从终端用户角度观察程序性能状况。

一旦测试完毕，LoadRunner 收集汇总所有的测试数据，并提供高级的分析和报告工具，以便迅速查找到性能问题并追溯缘由。使用 LoadRunner 的 Web 交易细节监测器，用

户可以了解到将所有的图像、框架和文本下载到每一网页上所需的时间。例如,这个交易细节分析机制能够分析是否因为一个大尺寸的图形文件或是第三方的数据组件造成应用系统运行速度减慢。另外,Web 交易细节监测器分解用于客户端、网络和服务器上端到端的反LoadRunner 支持广泛的协议,可以测试各种 IT 基础架构。

图 9-1 给出了 LoadRunner 的性能测试过程。LoadRunner 将性能测试过程分为计划测试、测试设计、创建 VU 脚本、创建测试场景、运行测试场景和分析结果 6 个步骤。

计划测试阶段主要进行测试需求的收集、典型场景的确定;测试设计阶段主要进行测试用例的设计;创建 VU 脚本阶段主要根据设计的用例创建脚本;创建测试场景阶段主要进行测试场景的设计和设置,包括监控指标的设定;运行测试场景阶段对已创建的测试场景进行测试,收集相应数据;分析结果阶段主要进行结果分析和报告工作。

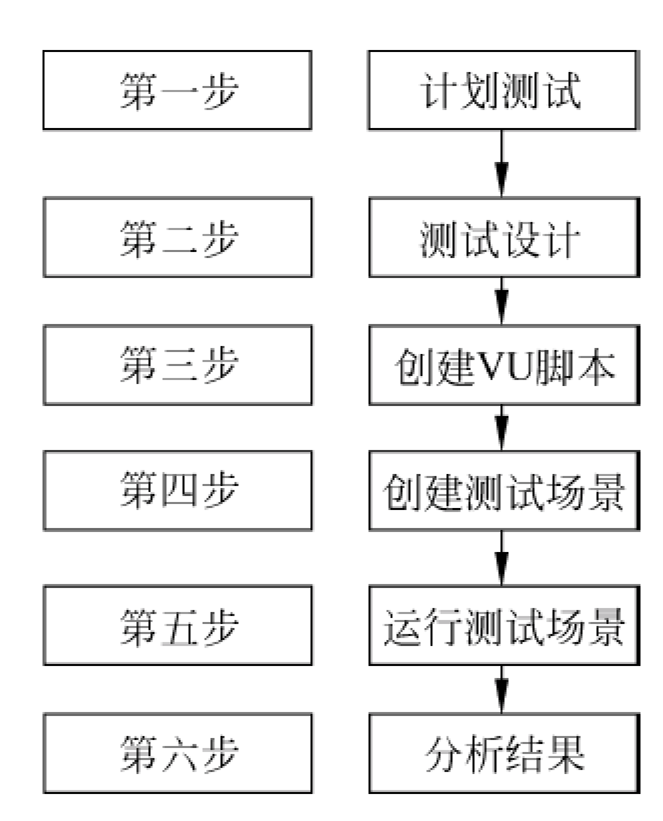


图 9-1 LoadRunner 的性能测试过程

LoadRunner 提供的这个性能测试过程已经涵盖了性能测试工作的大部分内容,但由于该过程过于紧密地与 LoadRunner 工具集成,没有兼顾使用其他工具,或是用户自行设计工具的需求,也不能称为是一个普适性的测试过程。

另外,LoadRunner 提供的该性能测试过程并未对计划测试阶段、测试设计阶段的具体行为、方法和目的进行详细描述,因此该方法最多只能称为"使用 LoadRunner 进行测试的过程",而不是一个适应性广泛的性能测试过程。

9.1.2 安装过程

(1) 运行 setup. exe,如图 9-2 所示。



图 9-2 运行 setup. exe

(2) 单击“安装”按钮,安装程序会自动检查所需组件是否都已安装,确定都安装后弹出如图 9-3 所示页面。

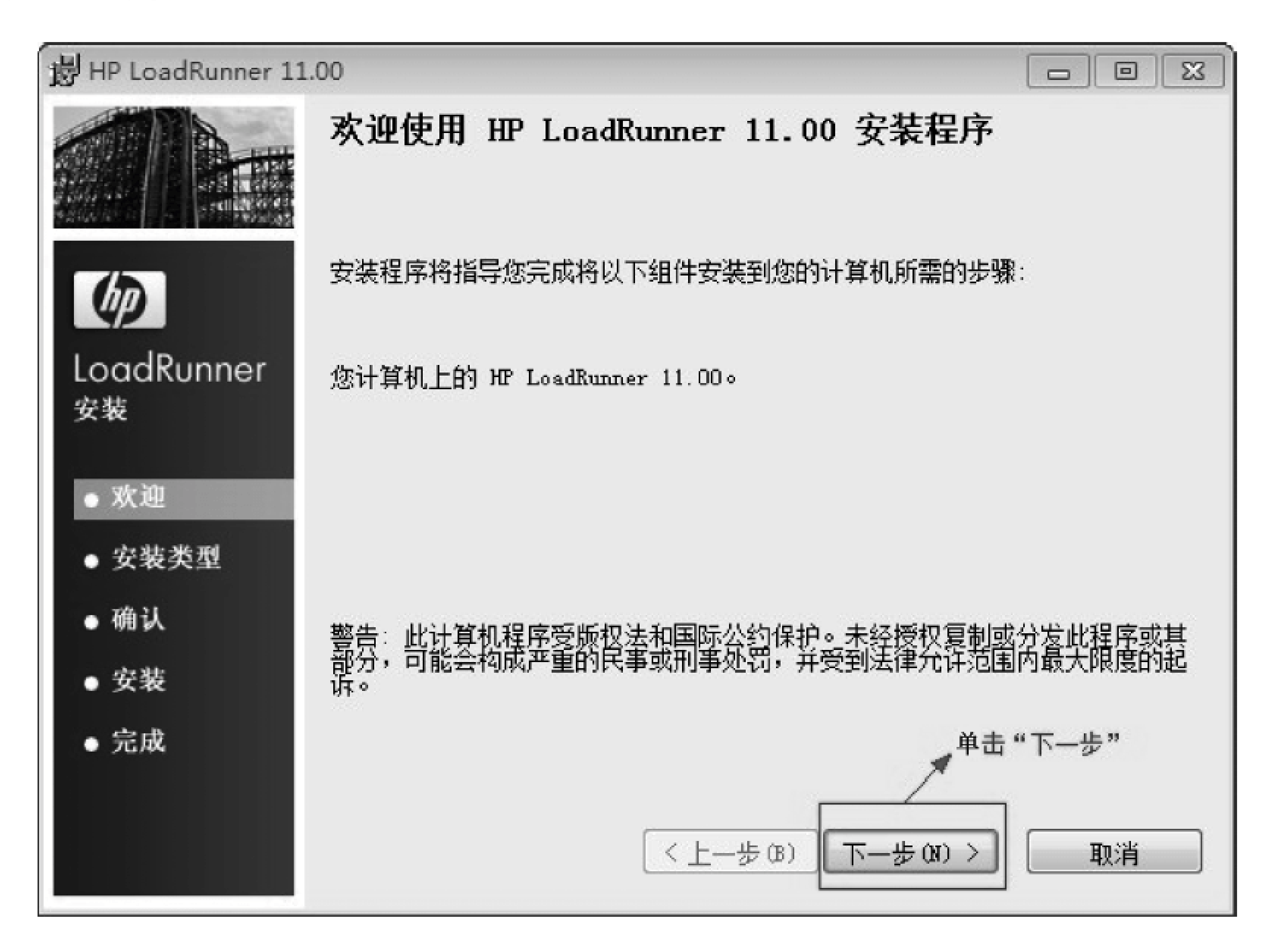


图 9-3　安装欢迎页面

(3) 选择“我同意”选项,单击“下一步”按钮,如图 9-4 所示。



图 9-4　许可协议

(4) 注意选择安装路径,单击“下一步”按钮,如图 9-5 所示。

(5) 确认安装,程序开始执行安装,单击“下一步”按钮,如图 9-6 所示。



图 9-5　选择安装路径



图 9-6　确认安装

(6) 显示程序执行安装的状态，如图 9-7 所示。

(7) 程序安装完成，单击“完成”按钮，如图 9-8 所示。

(8) 安装完成后，系统会自动打开 LoadRunner License Information 对话框，如图 9-9 所示。

此时，成功地安装了 LoadRunner 11。



图 9-7　程序执行安装的状态



图 9-8　安装完成

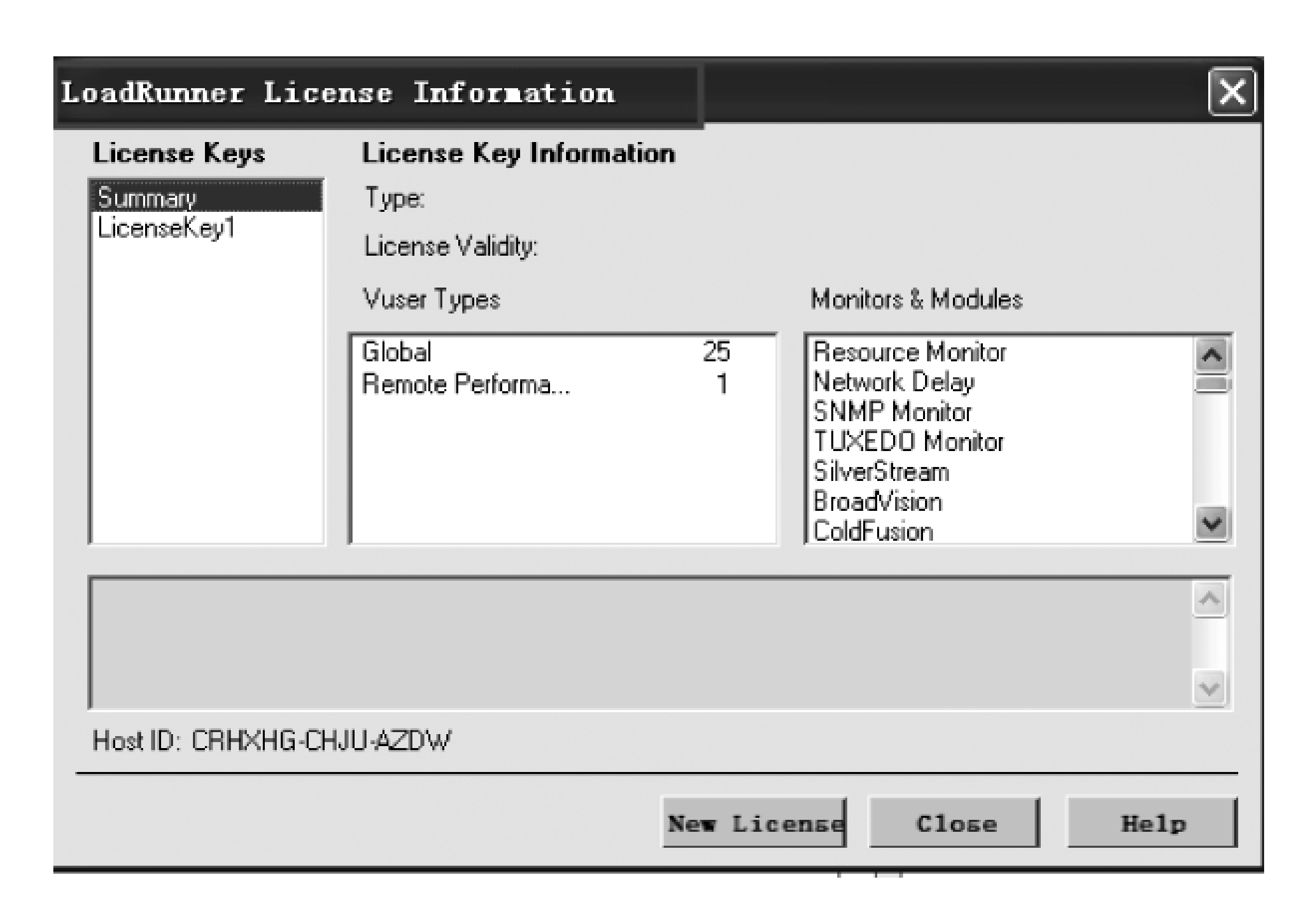


图 9-9　LoadRunner License Information 对话框

9.1.3 创建脚本

在 LoadRunner Launcher 窗格中,单击"创建/编辑脚本",打开 VuGen 起始页。

在欢迎使用 Virtual User Generator 区域中,单击"新建脚本"按钮,打开"新建虚拟用户"对话框,显示"新建单协议脚本"选项,如图 9-10 所示。



图 9-10 "新建单协议脚本"选项

协议是客户端用来与系统后端进行通信的语言。HP Web Tours 是一个基于 Web 的应用程序,因此将创建一个 Web Vuser 脚本。确保"类别"是所有协议。VuGen 将列出适用于单协议脚本的所有可用协议。向下滚动列表,选择 Web (HTTP/HTML)并单击创建,创建一个空白 Web 脚本。

空白脚本以 VuGen 的向导模式打开,同时左侧显示任务窗格。如果没有显示任务窗格,单击工具栏上的任务按钮。如果"开始录制"对话框自动打开,单击取消。

VuGen 的向导指导用户逐步完成创建脚本并使其适应测试环境的过程。任务窗格中列出脚本创建过程中的各个步骤或任务。如图 9-11 所示,在用户执行各个步骤的过程中,VuGen 将在窗口的主要区域显示详细说明和指示信息。

可以自定义 VuGen 窗口来显示或隐藏各个工具栏。要显示或隐藏工具栏,选择"视图"→"工具栏",并选中/不选中目标工具栏旁边的复选标记。

单击"录制"窗格中的"录制应用程序",如图 9-12 所示。

也可以选择 Vuser→"开始录制"或者单击页面顶部工具栏中的"开始录制"按钮,打开"开始录制"对话框,如图 9-13 所示。

在"URL 地址"框中输入要录制的地址。在"录制到操作"框中,选择 Action。单击"确定"按钮。

录制结束后在浮动工具栏上单击"停止"以停止录制,并保存结果。

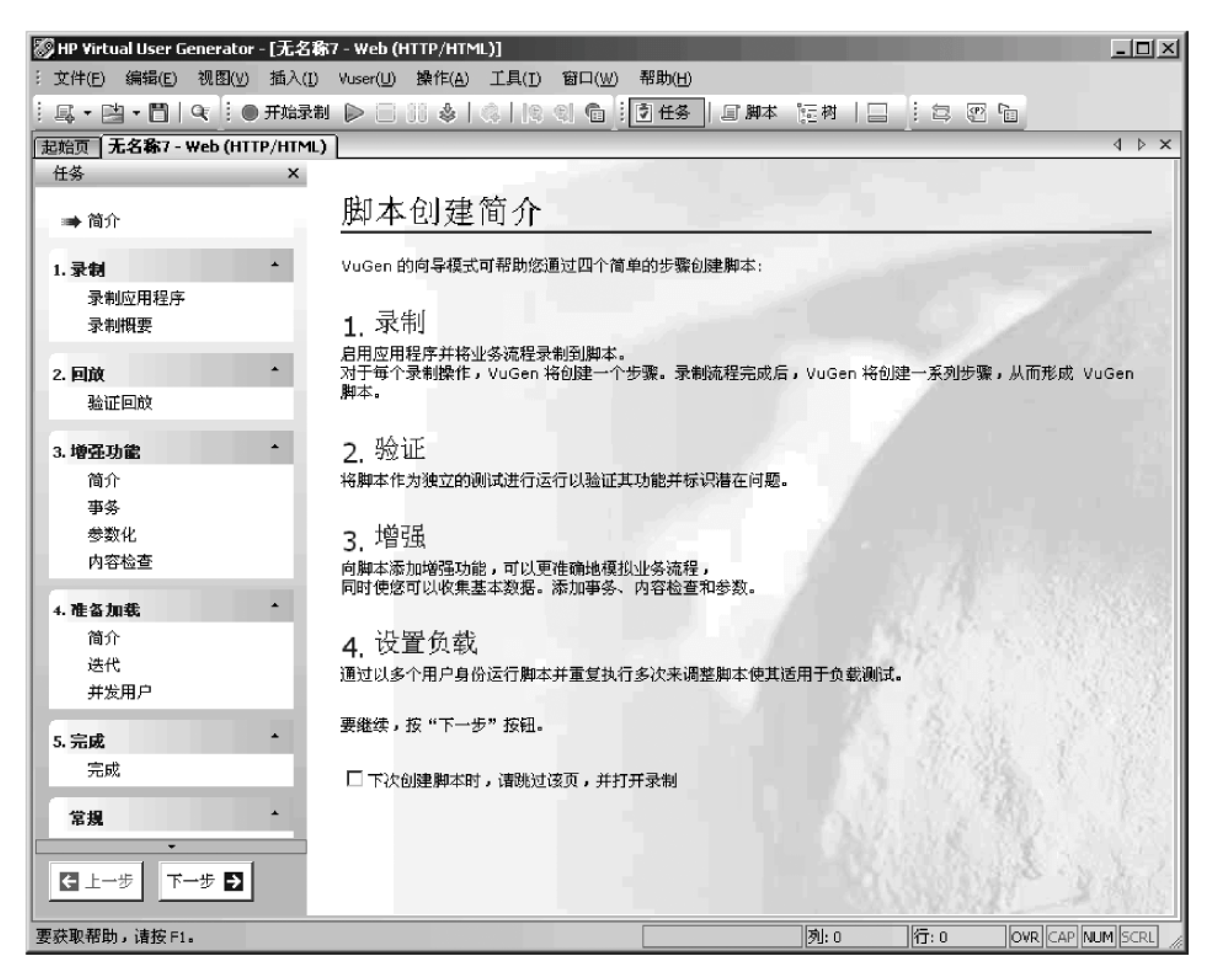


图 9-11 脚本创建简介



图 9-12 录制简介



图 9-13 “开始录制”对话框

Vuser 脚本生成时会打开“代码生成”弹出窗口。然后 VuGen 向导会自动执行任务窗格中的下一步,并显示关于录制情况的概要信息。

如图 9-14 所示,“录制概要”包含协议信息以及会话期间创建的一系列操作。VuGen 为录制期间执行的每个步骤生成一个快照,即录制期间各窗口的图片。这些录制的快照以缩略图的形式显示在右窗格中。如果由于某种原因要重新录制脚本,可单击页面底部的“重新录制”按钮。

至此脚本录制完毕。



图 9-14 录制概要

9.1.4 负载测试

打开 HP LoadRunner。选择“开始”→“程序”→HP LoadRunner→LoadRunner,打开 HP LoadRunner 窗口。

在 LoadRunner Launcher 窗格中单击“运行负载测试”，打开 HP LoadRunnerController，如图 9-15 所示。

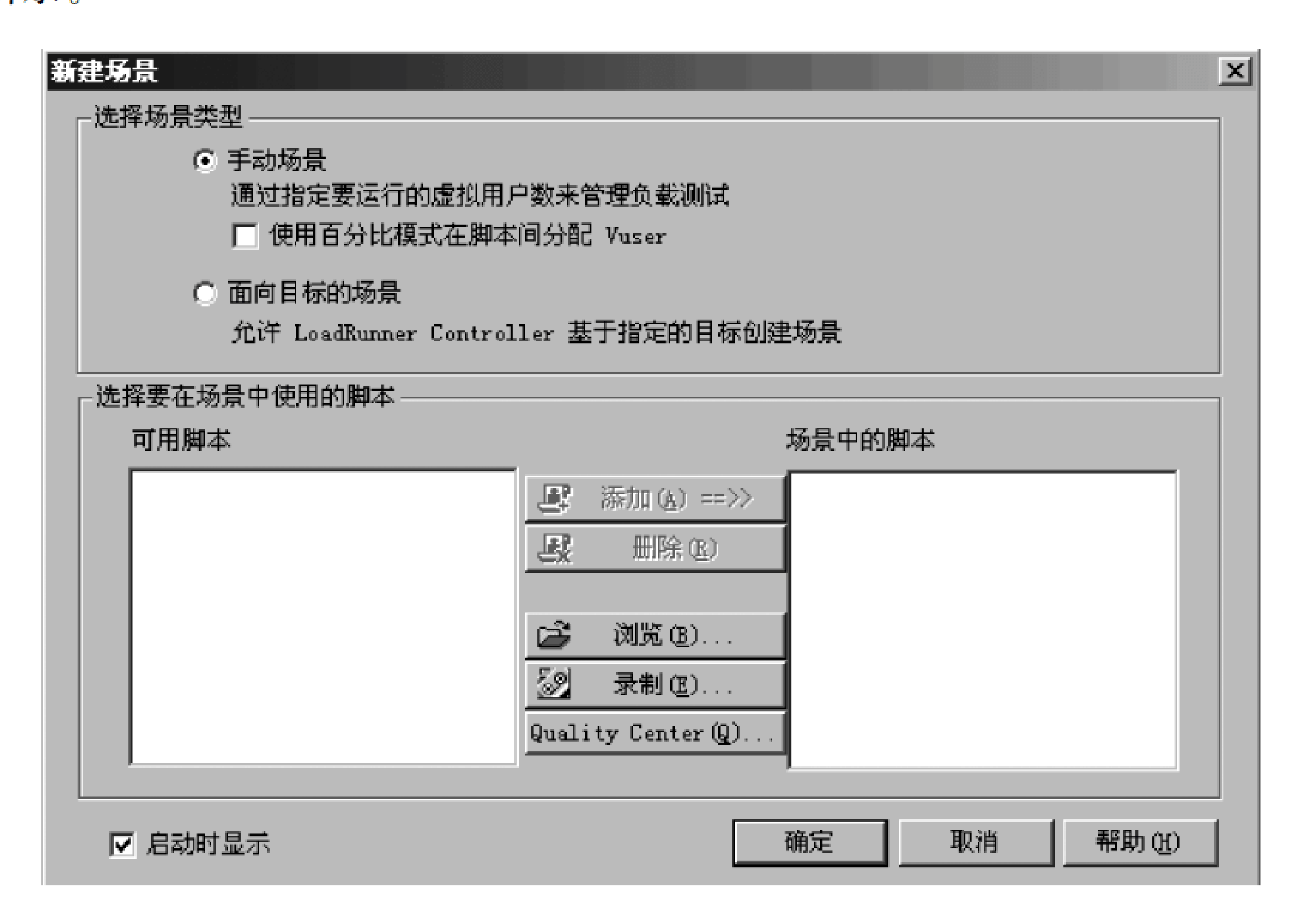


图 9-15 新建场景

选择录制好的脚本，单击“确定”按钮，打开 Controller 窗口，如图 9-16 所示。

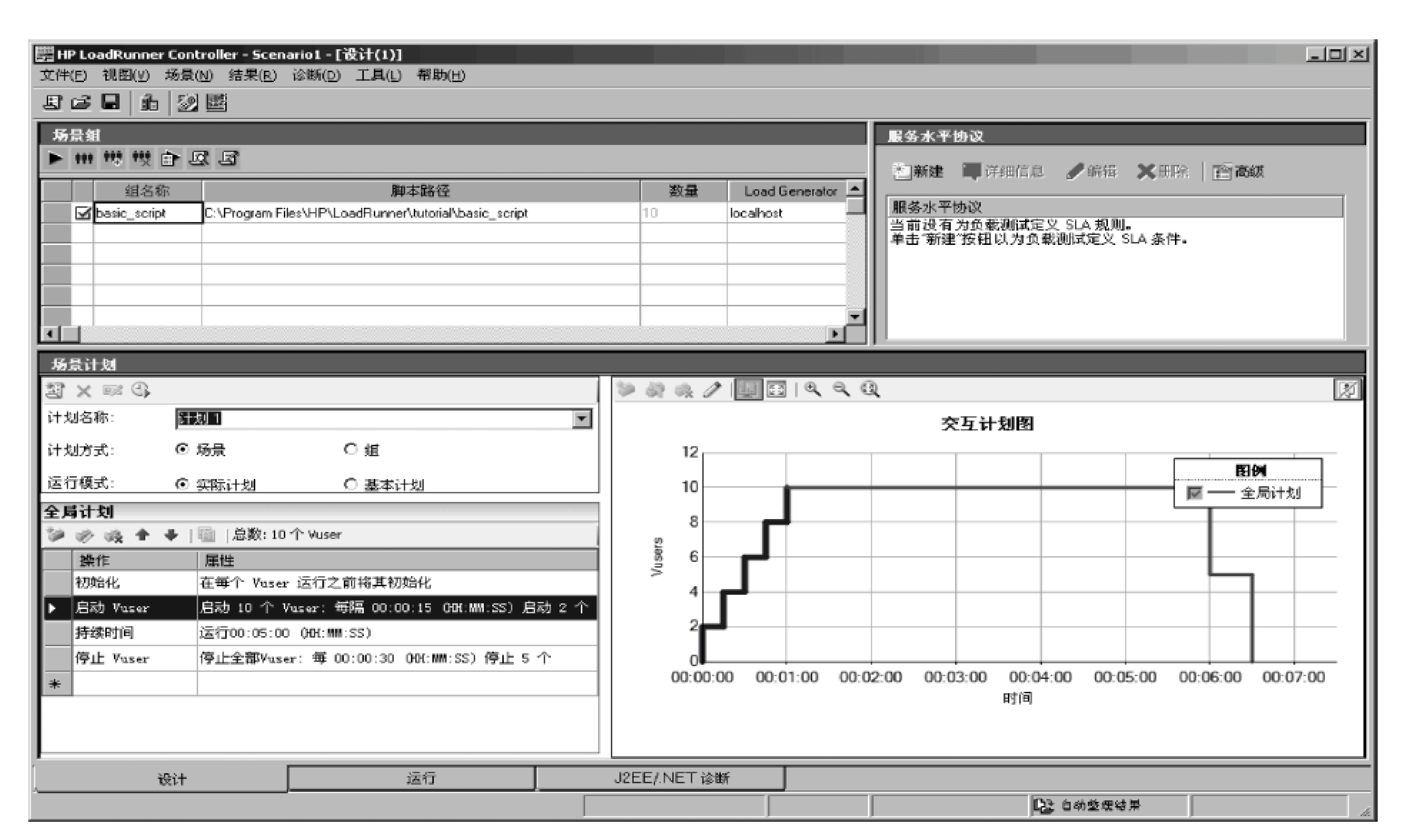


图 9-16 Controller 窗口

在“场景组”窗格中配置 Vuser 组，可以创建代表系统中典型用户的不同组，指定运行的 Vuser 数目以及运行时使用的计算机。

设计负载测试场景时，可以为性能指标定义目标值或服务水平协议(SLA)。运行场景时，LoadRunner 收集并存储与性能相关的数据。分析运行情况时，Analysis 将这些数据与

SLA 进行比较,并为预先定义的测量指标确定 SLA 状态。

从 Controller 菜单中选择"文件"→"打开",并打开录制好的脚本,将打开 LoadRunner Controller 的"设计"选项卡,如图 9-17 所示。

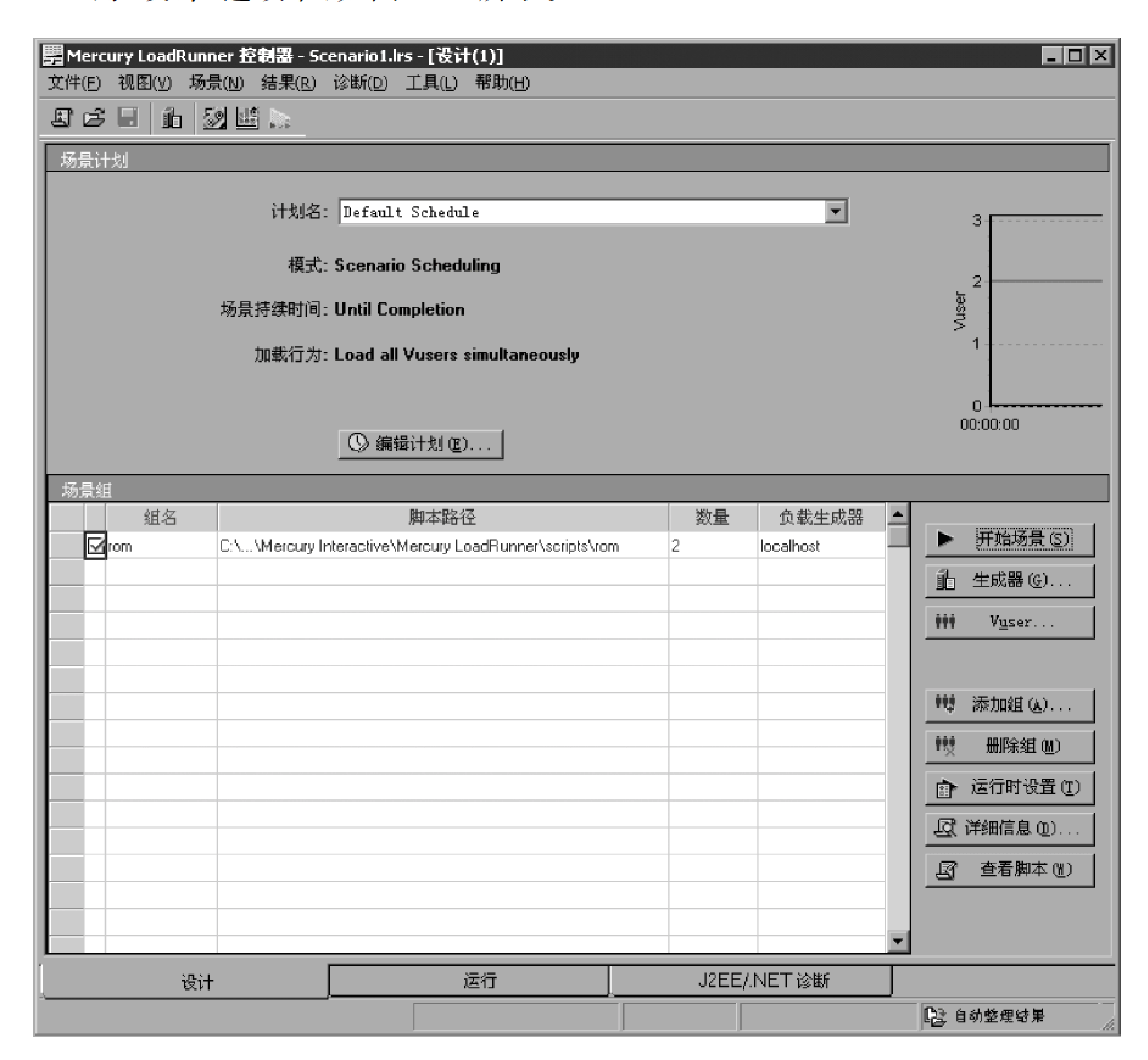


图 9-17 "设计"选项卡

此时,可以准备运行测试了。

1. 运行时设置

(1) 打开"运行时设置"。

确保显示"任务"窗格(如果未单击"任务"按钮)。在"任务"窗格中单击"验证回放"。在说明窗格中的标题"运行时设置"下单击"打开运行时设置"超链接。还可以按 F4 键或单击工具栏中的"运行时设置"按钮,打开"运行时设置"对话框,如图 9-18 所示。

(2) 打开"运行逻辑"设置,如图 9-19 所示。选择"运行逻辑"节点。

(3) 进行"步"设置,如图 9-20 所示。

通过此节点可以控制迭代之间的时间。可以将此时间指定为随机时间。这将准确模拟用户在操作之间等待的实际时间设置,但在随机时间间隔下,用户看不到实际用户在重复操作之间等待恰好为 60s 的情况。

选择第三个选项并选择 60.00~90.00s 之间的随机时间间隔。



图 9-18　“运行时设置”对话框

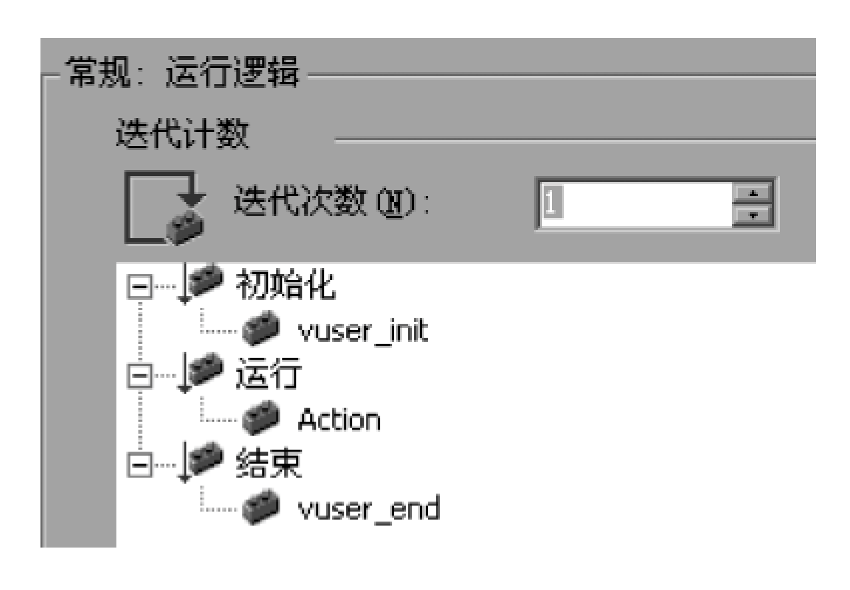


图 9-19　选择“运行逻辑”节点

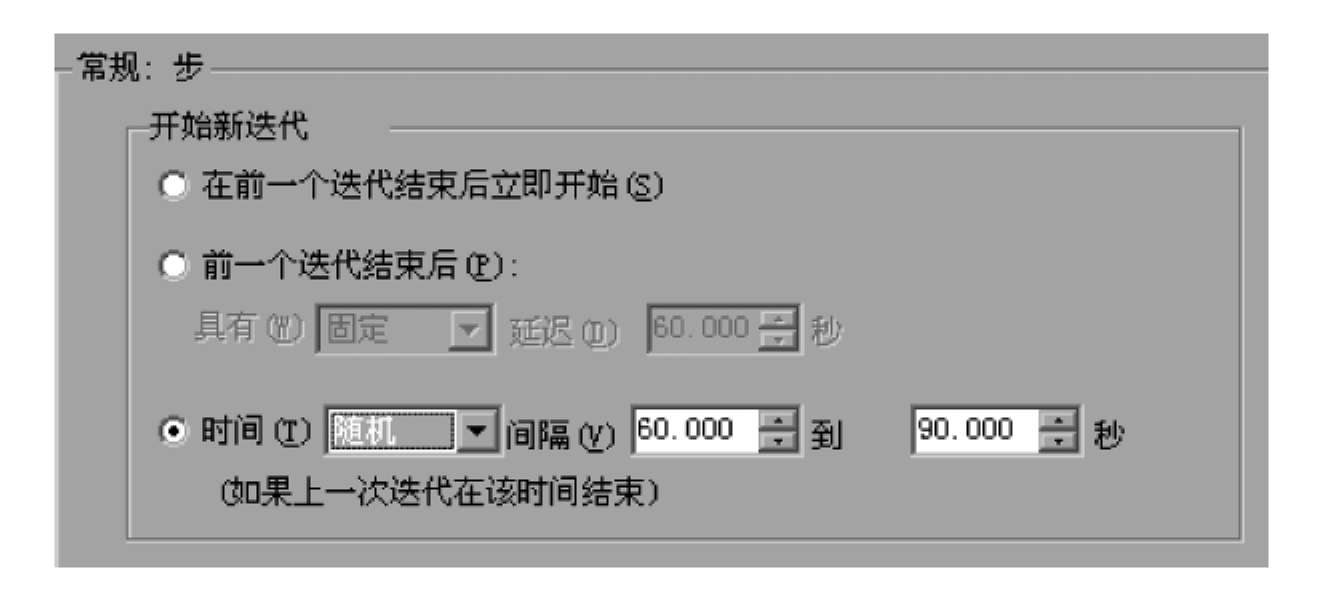


图 9-20　“步”设置

（4）设置“日志”设置，如图 9-21 所示。



图 9-21　“日志”设置

“日志”设置指示运行测试时要记录的信息详细级别。开发期间，出于调试目的，可以选择启用某级别的日志记录，但验证脚本可以正常工作后，仅可以启用或禁用错误日志记录。选择“扩展日志”并启用“参数替换”。

(5) 查看“思考时间”设置，如图 9-22 所示。

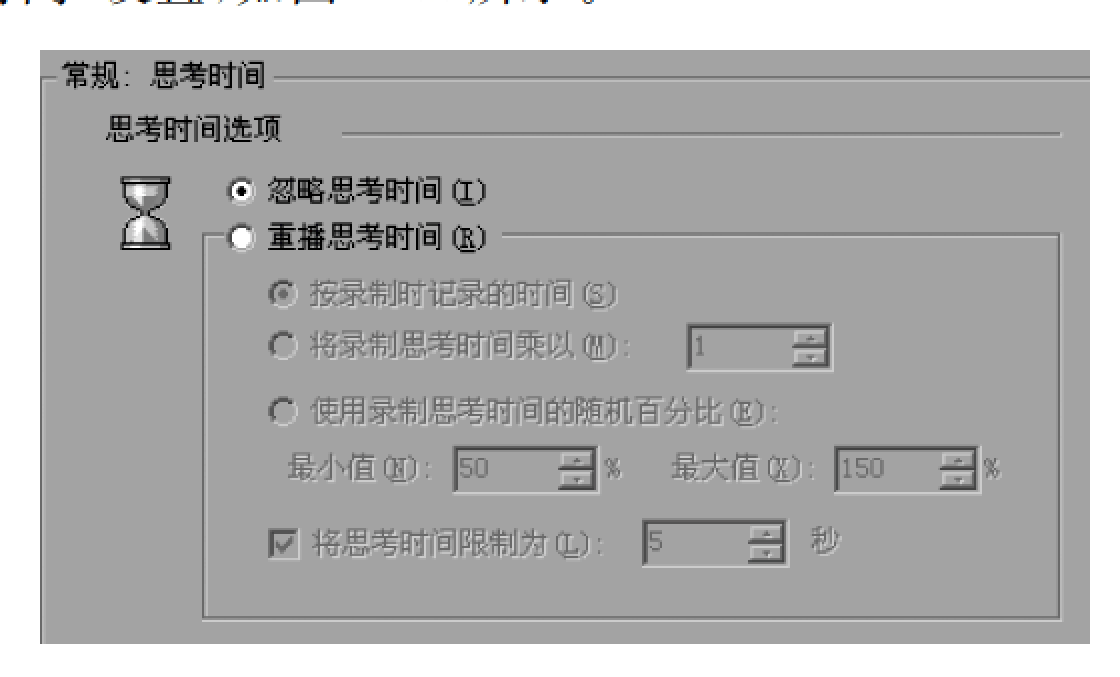


图 9-22 “思考时间”设置

请勿进行任何更改，可通过 Controller 设置思考时间。记住，在 VuGen 中运行脚本时，由于脚本不包括思考时间，因此脚本将快速运行。

(6) 单击“确定”按钮关闭“运行时设置”对话框。

2. 运行负载测试

单击“启动场景”按钮。如图 9-23 所示，将显示 Controller 运行视图，Controller 将开始运行场景。在“场景组”窗格中，可以看到 Vuser 逐渐开始运行并在系统上生成负载。可以在联机图上看到服务器对 Vuser 操作的响应度。



图 9-23 运行场景

9.1.5　脚本运行状态

1. 检查性能图

如图 9-24 所示，“正在运行 Vuser -整个场景”图显示在指定时间运行的 Vuser 数；“事务响应时间-整个场景”图显示完成每个事务所用的时间；“每秒点击次数-整个场景”图显示场景运行期间 Vuser 每秒向 Web 服务器提交的点击次数(HTTP 请求数)；“Windows 资源-最后一个 60 秒”图显示场景运行期间评测的 Windows 资源。

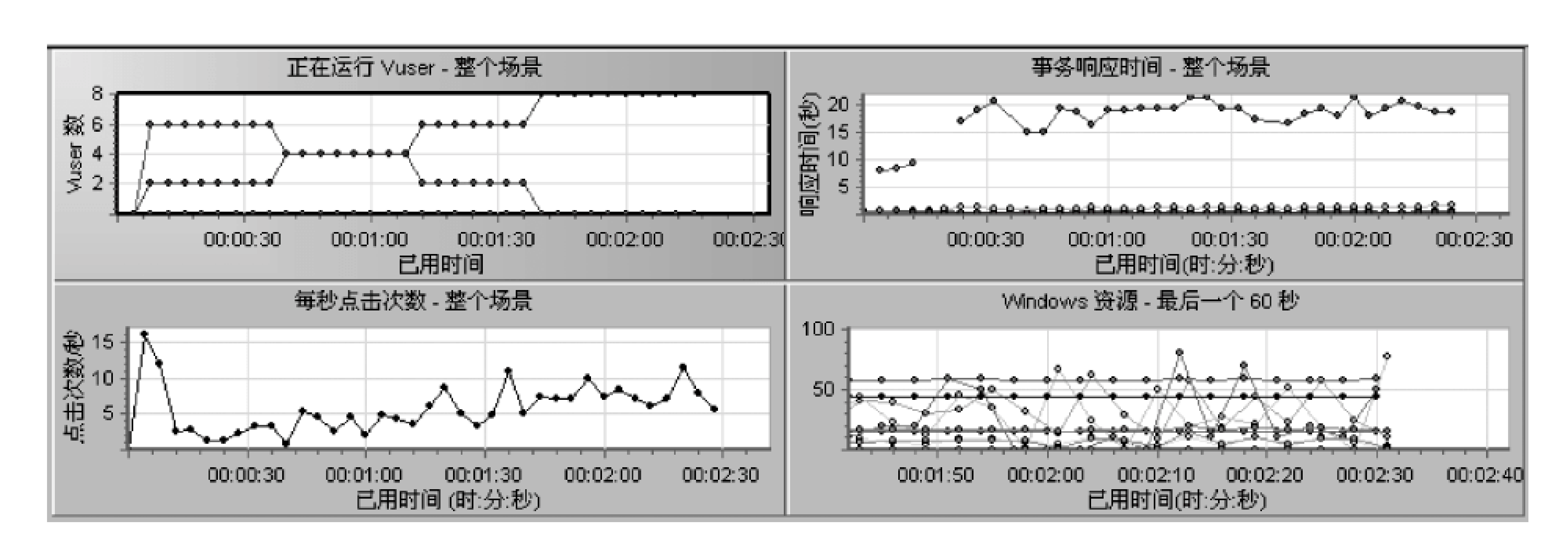


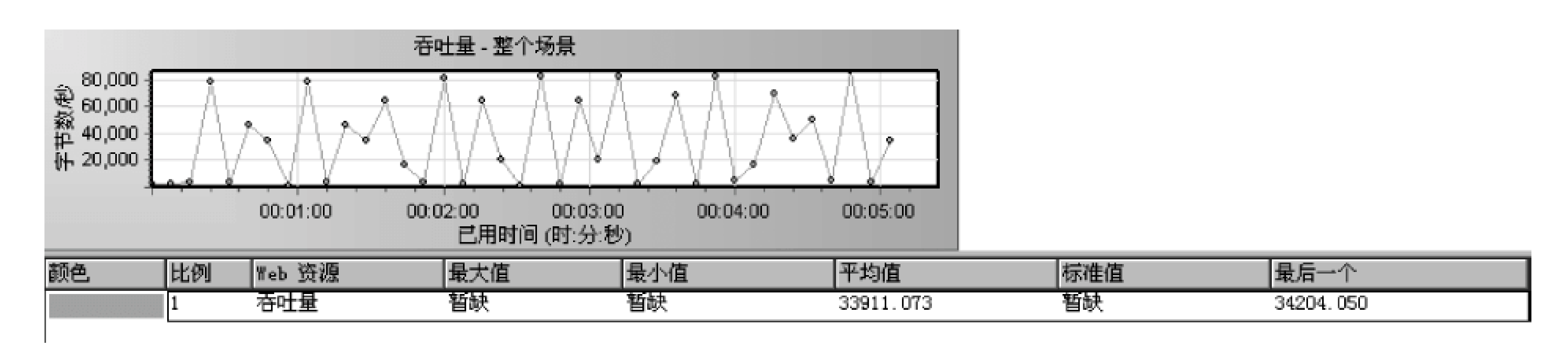
图 9-24　性能图

2. 突出显示单个测量值

双击“Windows 资源”图将其放大。注意每个测量值都显示在图例中用不同颜色标记的行中，每行对应图中与之颜色相同的一条线。选中一行时，图中的相应线条将突出显示，反之则不突出显示。再次双击图将其缩小。

3. 查看吞吐量信息

如图 9-25 所示，选择可用图树中的吞吐量图，将其拖放到图查看区域。“吞吐量”图中的测量值显示在画面窗口和图例中。“吞吐量”图显示 Vuser 每秒从服务器接收的数据总量(以字节为单位)。可以将此图与“事务响应时间”图比较，查看吞吐量对事务性能的影响。如果随着时间的推移和 Vuser 数目的增加，吞吐量不断增加，说明带宽够用。如果随着 Vuser 数目的增加，吞吐量保持相对平稳，可以认为是带宽限制了数据流量。



颜色	比例	Web 资源	最大值	最小值	平均值	标准值	最后一个
	1	吞吐量	暂缺	暂缺	33911.073	暂缺	34204.050

图 9-25　吞吐量信息

模拟用户时，应该能够实时查看用户的操作，确保执行正确的步骤。通过 Controller，可以使用运行时查看器实时查看操作。

4. 查看 Vuser 信息

要直观地查看 Vuser 的操作,执行以下操作。

单击 Vuser 按钮,打开 Vuser 窗口,如图 9-26 所示。

Vuser (8)

travel_agent　所有 Vuser

ID	状态	脚本	Load Generator	已用时间
1	正在运行	basic_script	localhost	00:00:37
2	正在运行	basic_script	localhost	00:00:37
5	正在运行	basic_script	localhost	00:00:06
6	正在运行	basic_script	localhost	00:00:06
7	关闭	basic_script	localhost	
8	关闭	basic_script	localhost	
9	关闭	basic_script	localhost	
10	关闭	basic_script	localhost	

运行(R)　逐渐停止(G)　停止(S)　重置(E)　详细信息(D)...　添加 Vuser(A)...　帮助(H)　关闭(C)

图 9-26　Vuser 窗口

状态列显示每个 Vuser 的状态。在上例中,可以看到有 4 个正在运行的 Vuser 和 4 个已经关闭的 Vuser。计划程序中的启动 Vuser 操作指示 Controller 每次释放两个 Vuser。随着场景的运行,将继续每隔 30s 向组中添加两个 Vuser。

从 Vuser 列表中选择一个正在运行的 Vuser。单击 Vuser 工具栏上的显示选定的 Vuser 按钮,打开运行时查看器并显示所选 Vuser 当前执行的操作。当 Vuser 执行录制的脚本中所包含的各个步骤时,窗口不断更新。单击 Vuser 工具栏上的“隐藏选定的 Vuser”按钮,关闭“运行时查看器”日志。

5. 查看用户操作信息

对于正在运行的测试,要检查测试期间各个 Vuser 的进度,可以查看包含 Vuser 操作文本概要信息的日志文件。

要查看事件的文本概要信息,执行以下操作。

在 Vuser 窗口中选择一个正在运行的 Vuser,单击“显示 Vuser 日志”按钮,Vuser 日志窗口打开,如图 9-27 所示。

日志中包含与 Vuser 操作对应的消息。例如,在上面的窗口中,消息 Virtual UserScript started 说明场景已启动。滚动到日志底部,查看为所选 Vuser 执行的每个操作添加的新消息。

关闭 Vuser 日志窗口和 Vuser 窗口。

6. 查看运行状态

“场景状态”窗格显示场景的整体状况,如图 9-28 所示。



图 9-27 Vuser 日志窗口

场景状态	正在运行
运行 Vuser	3
已用时间	00:01:58 (hh:mm:ss)
每秒点击次数	0.17 (最后 60秒)
通过的事务	76
失败的事务	0
错误	0

图 9-28 “场景状态”窗格

单击“场景状态”窗格中通过的事务，查看事务的详细信息列表。打开“事务”对话框，可以看到 Vuser 操作的详细信息，如图 9-29 所示。

事务

名称	TPS	通过	失败	停止
S01_T02_LogIn	0.1	17	0	0
S01_T03_SearchFlight	0.2	17	0	0
S01_T07_SignOff	0.2	17	0	0
vuser_end_Transaction	0.2	17	0	0
S01_T05_PaymentDetail	0.2	17	0	0
Action_Transaction	0.2	17	0	0
vuser_init_Transaction	0.1	17	0	0
S01_T04_BookFlight	0.2	17	0	0
S01_T01_HomePage	0.1	17	0	0
S01_T06_CheckItinerary	0.2	17	0	0

关闭(C) 帮助(H)

图 9-29 Vuser 操作的详细信息

9.2 安全测试工具

9.2.1 Rational AppScan 原理及简介

Rational AppScan 是对 Web 应用和 Web Services 进行自动化安全扫描的黑盒工具，它不但可以简化企业发现和修复 Web 应用安全隐患的过程(这些工作以往都是由人工进行，成本相对较高，但是效率却非常低下)，还可以根据发现的安全隐患，提出针对性的修复建议，并能形成多种符合法规、行业标准的报告，方便相关人员全面了解企业应用的安全状况。

AppScan包括测试版、Build版、标准版、企业版。标准版是一个单机版工具，个人可以利用它针对Web应用进行黑盒测试。

AppScan拥有一个庞大完整的攻击规则库，也称为特征库，通过在HTTP Request中插入测试用例的方法实现几百种应用攻击，再通过分析HTTP Response判断该应用是否存在相应的漏洞。特征库是可以随时添加的。它的扫描分为以下两个阶段。

阶段一：探测阶段。探测站点下有多少个Web页面，并列出来。

阶段二：测试阶段。针对探测到的页面，应用特征库实施扫描。扫描完毕，会给出一个漏洞的详细报告。

界面分为以下5大区域。

(1) 视图区；

(2) Web应用程序树状列表区；

(3) 结果列表区；

(4) 漏洞统计区；

(5) 漏洞详细信息区。

IBM Rational AppScan解决方案能够在Web开发、测试、维护、运营的整个生命周期中，帮助企业高效地发现、解决安全漏洞，最大限度地保证应用的安全性。

9.2.2 Rational AppScan应用举例

Web安全检测主要分为两大类，分别是白盒检测和黑盒检测。白盒工具通过分析应用程序源代码以发现问题，而黑盒工具则通过分析应用程序运行的结果来报告问题。Rational AppScan(以下简称AppScan)属于后者，它是业界领先的Web应用安全检测工具，提供了扫描、报告和修复建议等功能。

1. Rational AppScan安装

如图9-30所示安装完成后，启动AppScan应用程序。



图9-30 安装AppScan

2. Rational AppScan 主界面

如图 9-31 所示的主界面中，菜单栏涵盖了 AppScan 中的所有可用功能。工具条中为常用功能的快捷菜单，如开始扫描、扫描配置、扫描专家等。左上部网站导航视图：在扫描过程中 AppScan 会按照一定的层次组织显示站点结构图(默认是按照 URL 层次进行组织，用户可以在扫描配置中更改这一设置)。右上部安全问题显示视图：AppScan 将在此视图中列出检测到的所有安全缺陷。左下部安全问题汇总视图：AppScan 将在此视图中列出检测到的安全缺陷统计信息。右下部安全问题详细信息视图：此视图的内容与安全问题显示视图相关，用来显示某特定安全问题的详细信息，包括问题介绍、修复建议、测试数据等。

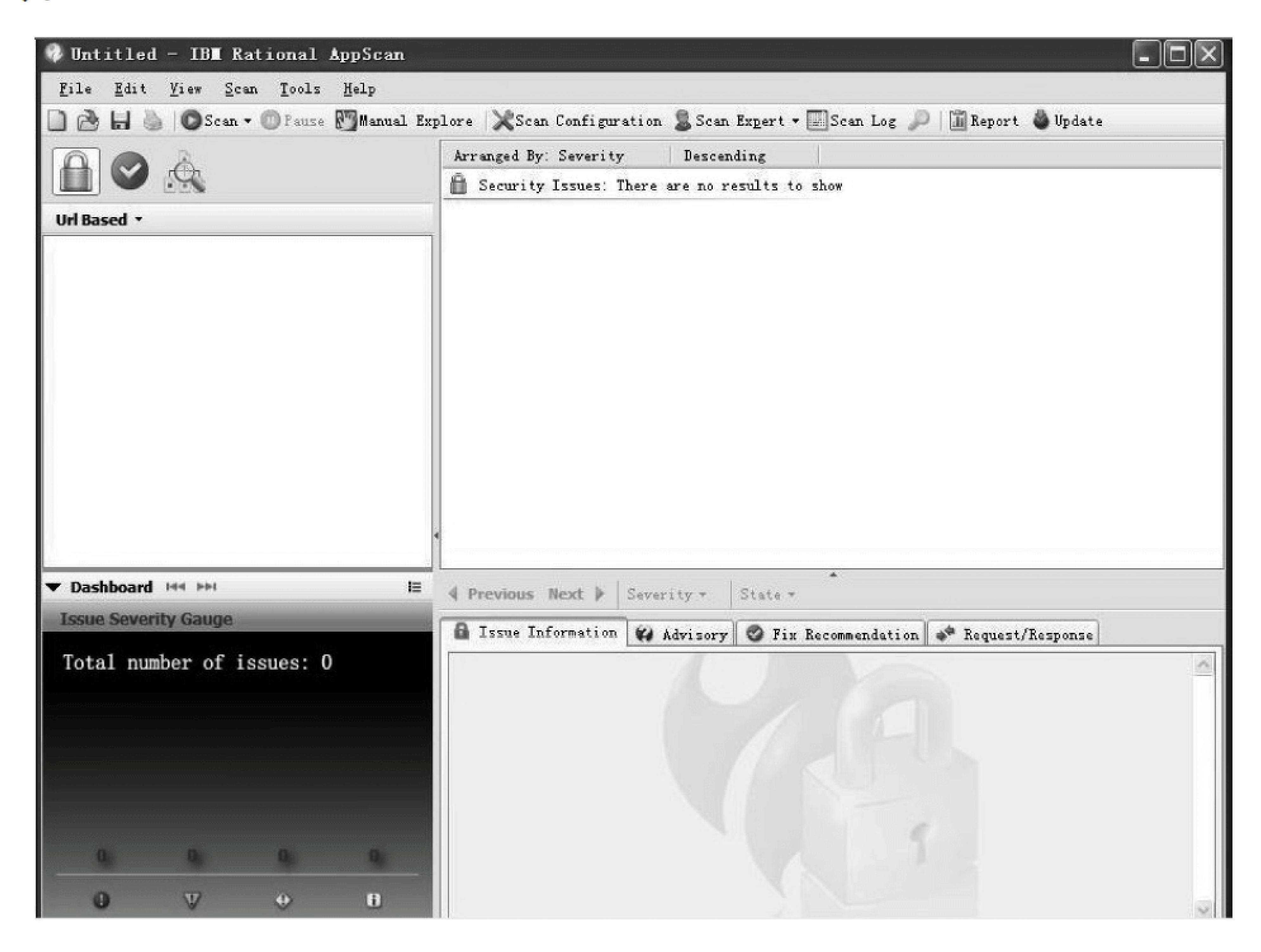


图 9-31　AppScan 主界面

3. Rational AppScan 配置扫描

首先单击“文件”菜单，在子菜单中选择“新建”，将弹出如图 9-32 所示对话框。对话框中列出了 AppScan 预先提供的常用扫描模板，使用模板可以大大简化扫描创建过程，提高工作效率；当然，用户也可以根据实际情况创建自定义模板。为了方便这里选择 demo. testfire. net，此模板是专门为测试站点 demo. testfire. net 而设立的。

然后需要选择安全扫描的类型：如果是 Web 站点扫描选择“Web 应用程序扫描”，如果是 Web 服务扫描选择“Web Service 扫描”。因为要对 demo. testfire. net 站点进行测试，所以选择“Web 应用程序扫描”，并单击“下一步”按钮，如图 9-33 所示。



图 9-32 新建扫描



图 9-33 扫描配置向导

下面是各种扫描参数的配置，包括 URL 和服务器、登录管理、测试策略 3 部分，因为 AppScan 模板已经为这些参数设置了合适的值，因此不需要做任何修改直接单击"下一步"按钮即可，如图 9-34 所示。

这里选择"自动"，并输入用户名和密码，单击"下一步"按钮，如图 9-35 所示。

测试策略使用默认的策略，单击"下一步"按钮，如图 9-36 所示。

选择"启动全面自动扫描"，勾选"完成'扫描配置向导'后启动'扫描专家'"选项，单击"完成"按钮，如图 9-37 所示。

进入扫描，扫描专家分析后生成扫描结果，如图 9-38 和图 9-39 所示。



图 9-34　扫描参数的配置



图 9-35　选择登录方法，并输入用户名和密码



图 9-36　测试策略



图 9-37 完成扫描配置向导

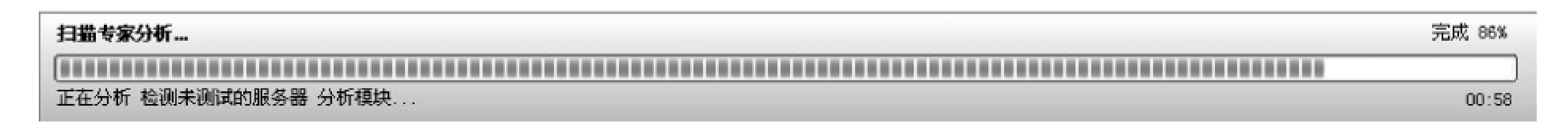


图 9-38 扫描专家分析

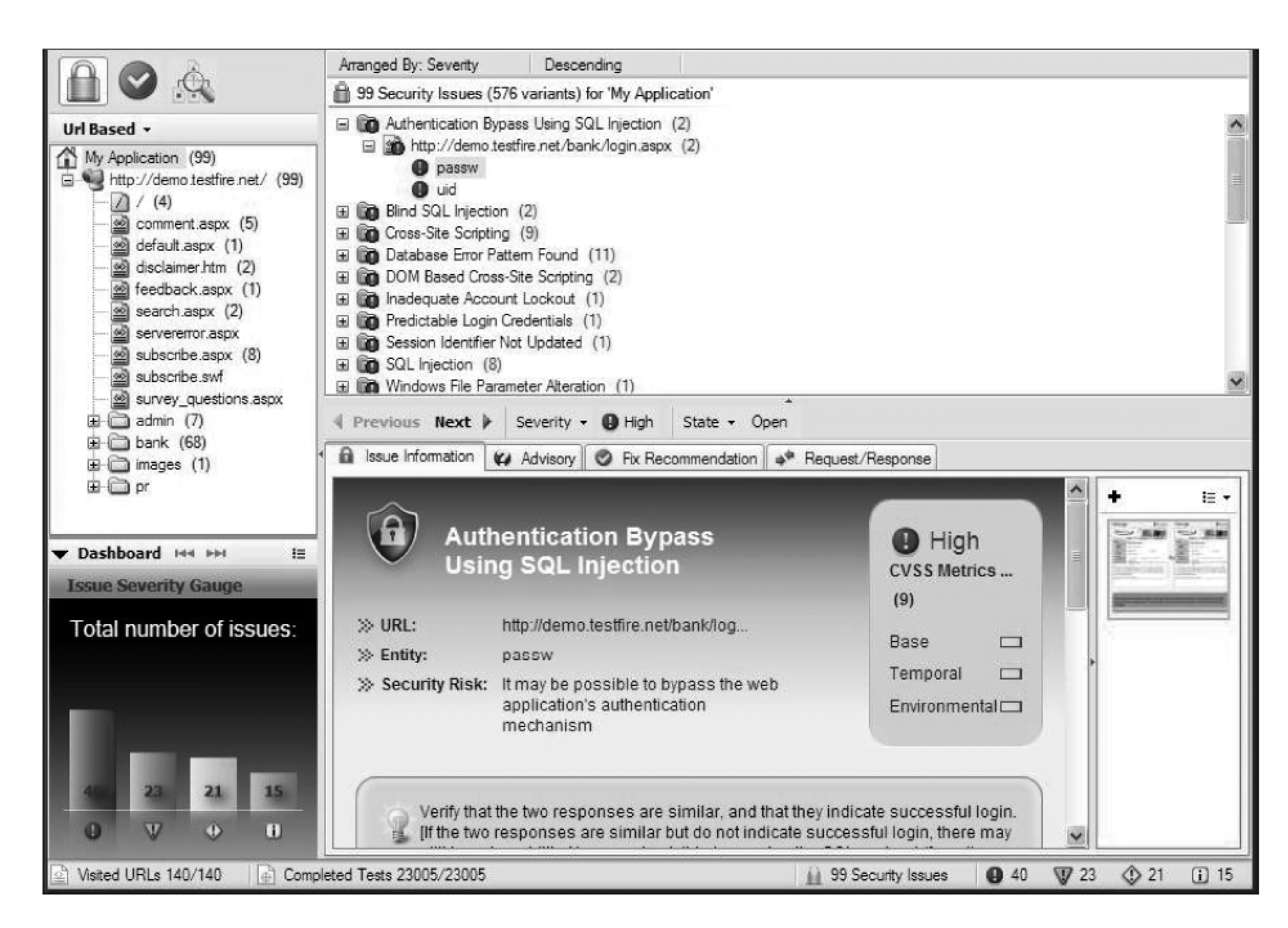


图 9-39 扫描结果

4. 扫描配置详解

使用 AppScan 对 demo.testfire.net 站点进行安全测试扫描时使用了模板，扫描参数都是预先设置的，读者可能对其意义及配置方法还不了解，下面将对它们进行详细介绍。扫描配置参数可以在创建扫描过程中设置，也可以在创建后进行设置，其配置过程和效果是完全相同的。如果选择的不是测试模板，则需要对 URL 和服务器、登录管理、测试策略和多步操作几部分进行配置。单击“扫描”下的“扫描配置”，对其进行配置，如图 9-40 所示。

1）配置 URL 和服务器

配置 URL 和服务器见图 9-40。

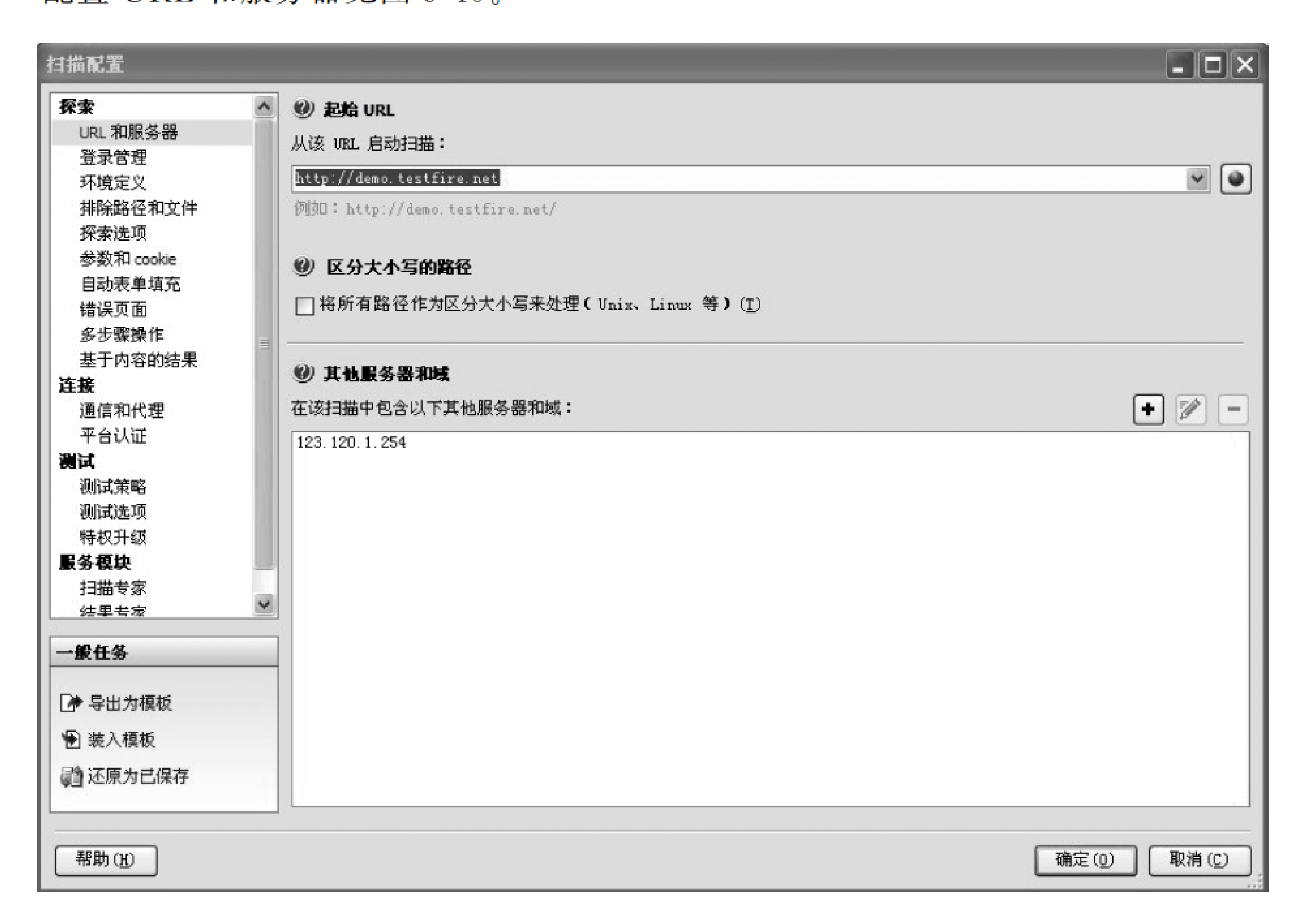


图 9-40 配置 URL 和服务器

2）配置登录管理

为了限制用户对站点部分功能的使用，绝大多数 Web 应用都实现了用户认证与授权，即通常所说的登录控制功能，非授权用户仅可以访问部分功能。AppScan 的安全测试是通过模拟用户对站点的访问来实现的，如果待测试站点有需要授权才能访问的内容，就必须让 AppScan 在测试过程中通过网站的认证。AppScan 提供登录管理来实现此功能，如图 9-41 所示。

记录(推荐)：推荐方式，需要用户手工录制登录过程；当 AppScan 发现需要登录站点时将自动重放录制过程实现登录。单击面板上的红色的“记录”按钮，弹出 AppScan 的内置浏览器，在此浏览器中转到登录页面，输入账户信息完成登录即可，AppScan 将自动记录登录过程。

因为 AppScan 会扫描所有探测到的需要测试的页面，包括退出页面，如果 AppScan 在测试中访问了退出页面就会造成用户会话终止，从而影响后续的测试。为防止此类情况的

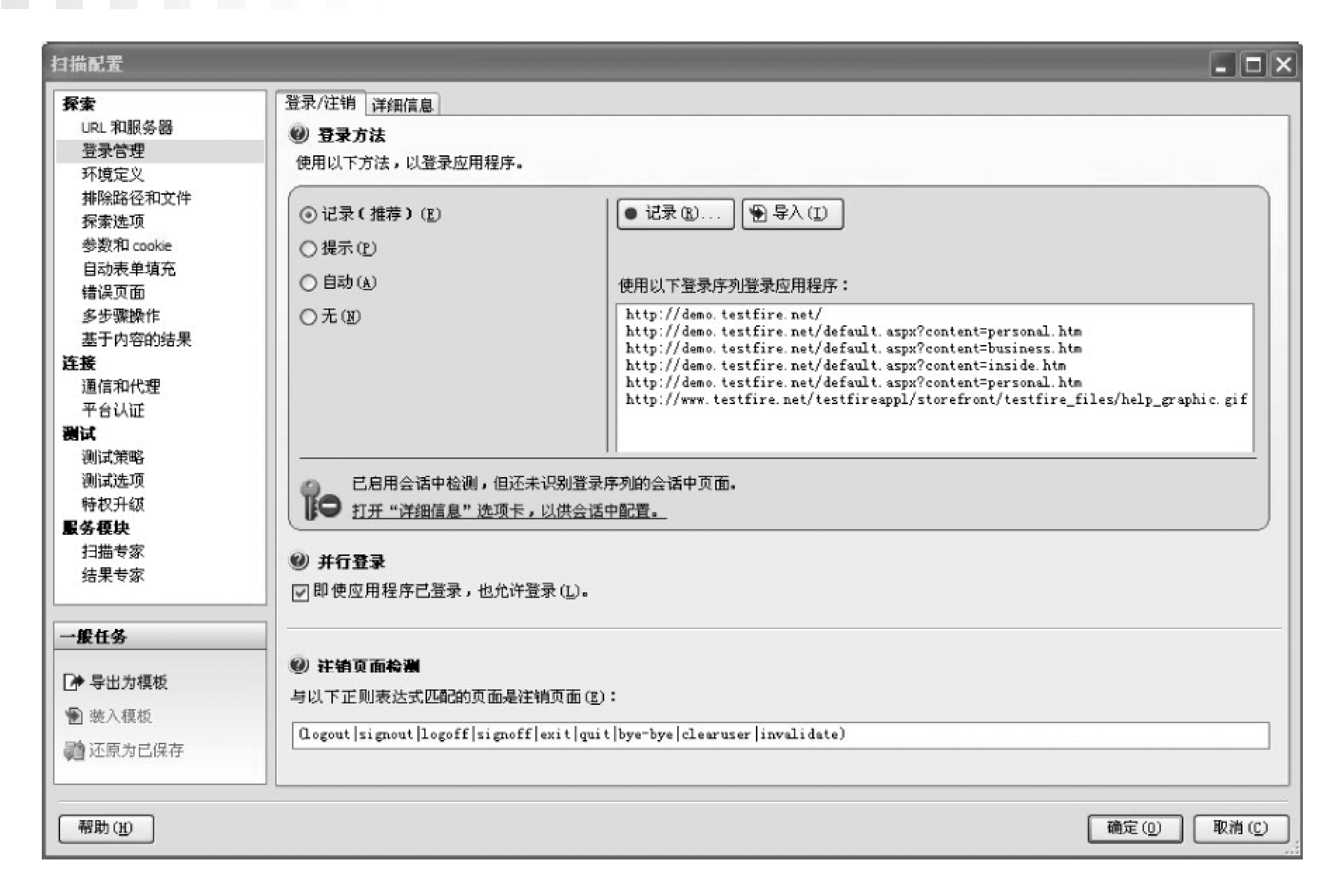


图 9-41 配置登录管理

发生就必须配置面板下方的"注销页面检测"，为登录页面设置正确的匹配模式，AppScan 再根据此参数的设置跳过对退出页面的测试。

3）配置测试策略

如图 9-42 所示，AppScan 会通过预先设定的安全规则创建测试用例，一条安全规则描述了某安全缺陷的信息，包括缺陷描述、修复建议、确认模式等。AppScan 集成了大规模的测试规则，默认情况下针对一个页面可能会生成成百上千个测试用例，但是这些用例中往往



图 9-42 配置测试策略

有很大一部分对于应用本身来说是没有任何意义的。为了减少测试冗余，增加测试的有效性就需要甄选符合应用实际情况的测试规则，例如针对 J2EE 站点的安全测试就不需要与 .NET 相关的测试规则，又如对安全性要求不高的站点（内部应用）只需要对高危险的安全缺陷进行探测而没必要测试所有。测试策略是对测试规则的管理，用户可以根据 Web 应用的实际项目情况选择合理的测试规则集，对选定的规则集可以通过 Export 的方式将其导出为策略文件，其他用户可以导入策略文件以保持组织内部策略规则的一致性。单击主配置面板左端的 Test Policy 打开测试策略视图，视图中列出了 AppScan 的所有测试规则，可以通过上面的分组下拉框对其按照不同方式进行分组以方便查看。

4）配置多步骤操作

如图 9-43 所示，对于复杂应用，要完成某项任务往往需要很多步骤，步骤之间有很强的顺序依赖关系。我们知道 AppScan 会测试所有发现的 URL，对这些 URL 的测试是随机的，它并不能保证对存在依赖关系的 URL 的测试是按照顺序的，这就可能造成 AppScan 首先访问后续步骤页面，从而影响安全测试结果。为了解决问题，AppScan 引入了多步骤序列的概念。单击主配置面板左边的多步骤操作打开多步骤操作视图，此视图中列出了所有已经创建的操作序列，可以将存在的序列导出成单独文件，也可以导入曾经被导出的序列文件；单击红色的按钮会弹出 AppScan 内置的浏览器，在此浏览器中用户只需像平时使用应用一样完成某项功能即可，关闭浏览器即完成序列脚本的录制。在测试过程中，AppScan 会将当前待测试的 URL 与录制的序列脚本中的 URL 进行比较，如果发现待测 URL 是某个序列中的一步，AppScan 会重放所有前置序列脚本以使对当前页面的访问合法。

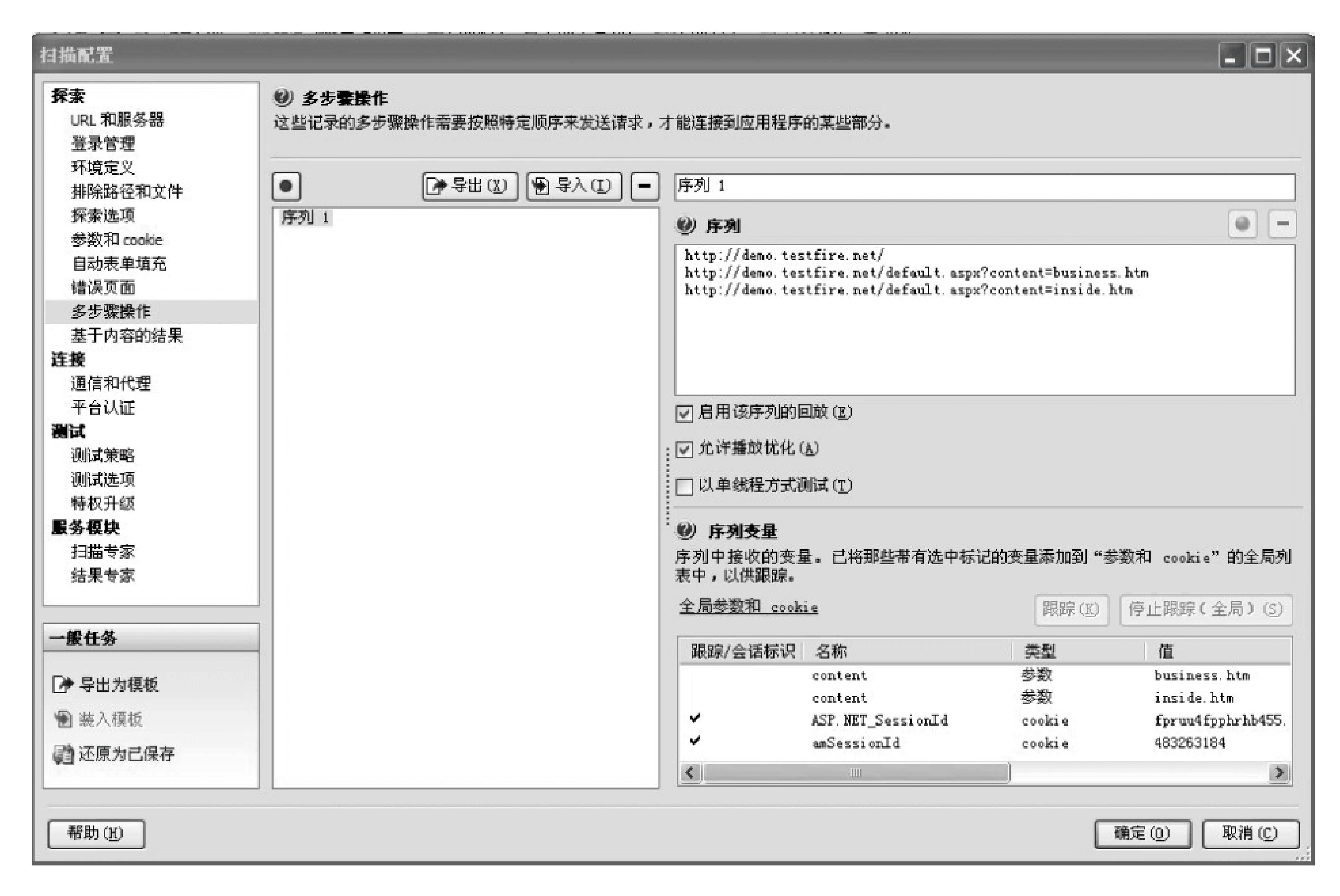


图 9-43　配置多步骤操作

9.2.3 Rational AppScan 扫描结果

Rational AppScan 扫描结果在 3 个视图中显示。通过视图选择上的按钮选择视图(默认为问题视图)。这 3 个视图中显示的数据会随选择的视图不同而改变,如表 9-1 所示。

表 9-1 3 个视图的显示内容

安全性问题	安全问题视图	从宏观到特定的请求/响应显示发现的实际问题 应用树:完整应用树。计数器显示每一项所发现的问题数 结果列表:显示所选树中节点的问题列表。显示问题的优先级别 详细资料栏:显示在结果列表上所选问题的顾问信息、修改建议、请求/响应
修复任务	修改工作视图	提供一个修改扫描中发现问题的详细修改意见表 应用树:完整应用树。计数器显示每一项所提供的修改意见数 结果列表:列出树中所选节点的意见列表。显示意见的优先级别 详细资料栏:显示结果栏中所选择修改意见和详细资料和问题的详细分析
应用程序数据	应用数据	显示检查执行期间的脚本参数、交互的 URL、访问的 URL、错误链接、过滤 URL、注释、Java 脚本和控件 应用树:完整应用树 结果列表:对结果列表栏上面可选列表进行过滤,以确定显示哪一项的详细信息 详细资料栏:结果列表中所选项的详细信息

严重等级如表 9-2 所示。

结果列表会显示应用树中所选择节点的问题,这些可以是:基本级(显示所有站点问题)、页面级(显示所有页面问题)、参数级(显示所有特定页面特定请求的问题)。

AppScan 给每一个发现的问题分配 4 个严重等级中的一个。

表 9-2 严重等级

	高严重级别
	中严重级别
	低严重级别
	报告安全问题 注意:这一类只适用于窗口问题。在修正窗口中所有比严重级别低的都属于低严重级别

9.3 案例分析

9.3.1 项目背景

1. 公司及产品简介

电讯盈科有限公司(简称电讯盈科,PCCW)是一家注册于中国香港的大型企业集团,是中国香港最大的电讯运营商。电讯盈科企业方案是电讯盈科集团信息技术和业务流程外包

服务的旗舰品牌。拥有3800名资深ICT专才,在中国香港、北京、上海、广州、深圳、武汉和西安等地均设有分支机构。

电讯盈科(PCCW)的维修管理信息系统(MMIS)产品已经成功应用于交通运输行业。

1)系统平台架构

PCCW MMIS系统平台架构如图9-44所示。

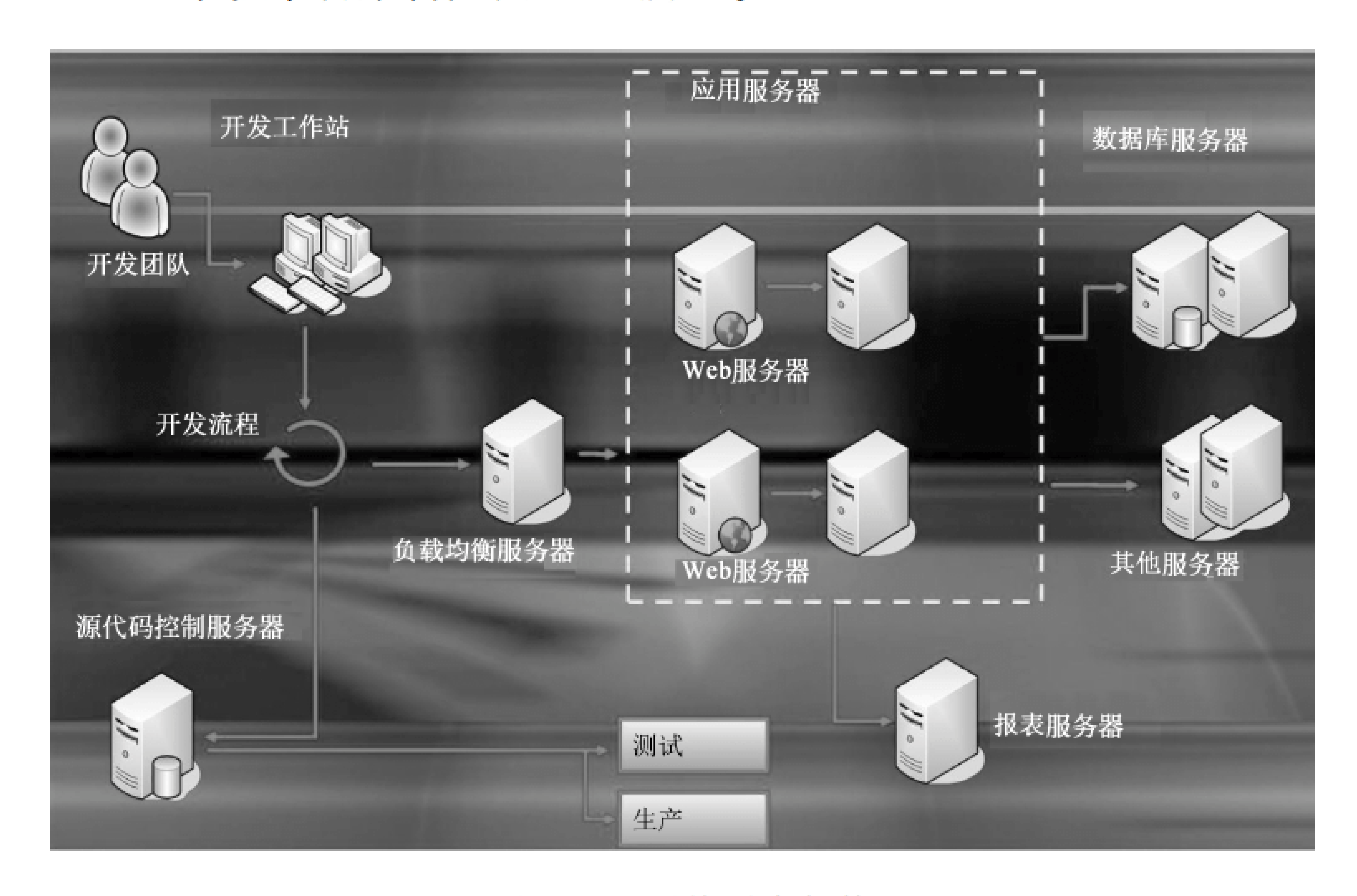


图9-44 系统平台架构

2)系统功能架构

PCCW MMIS系统功能架构如图9-45所示。

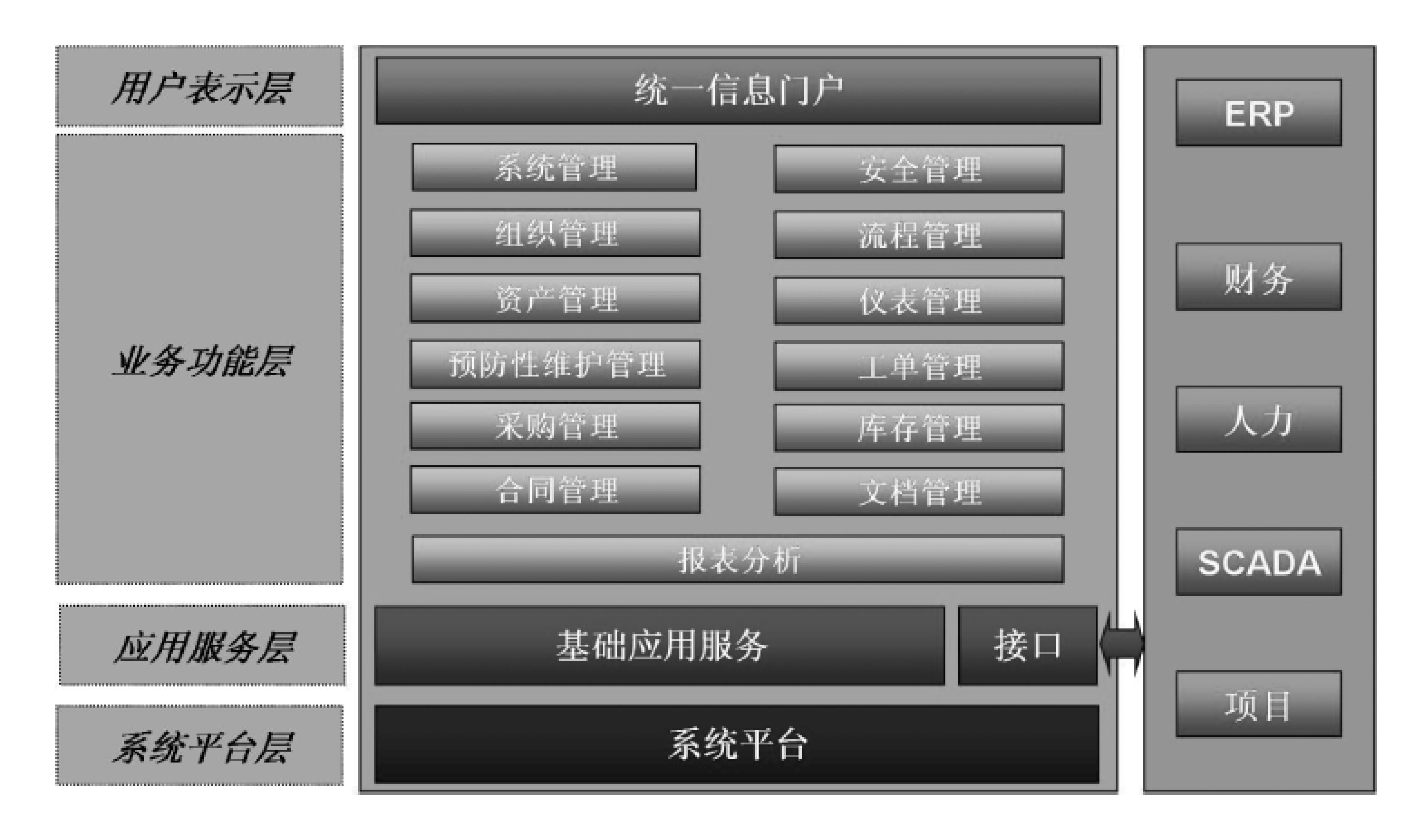


图9-45 系统功能架构

3)系统主要功能框架

PCCW MMIS系统主要功能框架如图9-46所示。

2. 主要功能

电讯盈科(PCCW)的维修管理信息系统(MMIS)通过优化维护资源,改善设备及员工

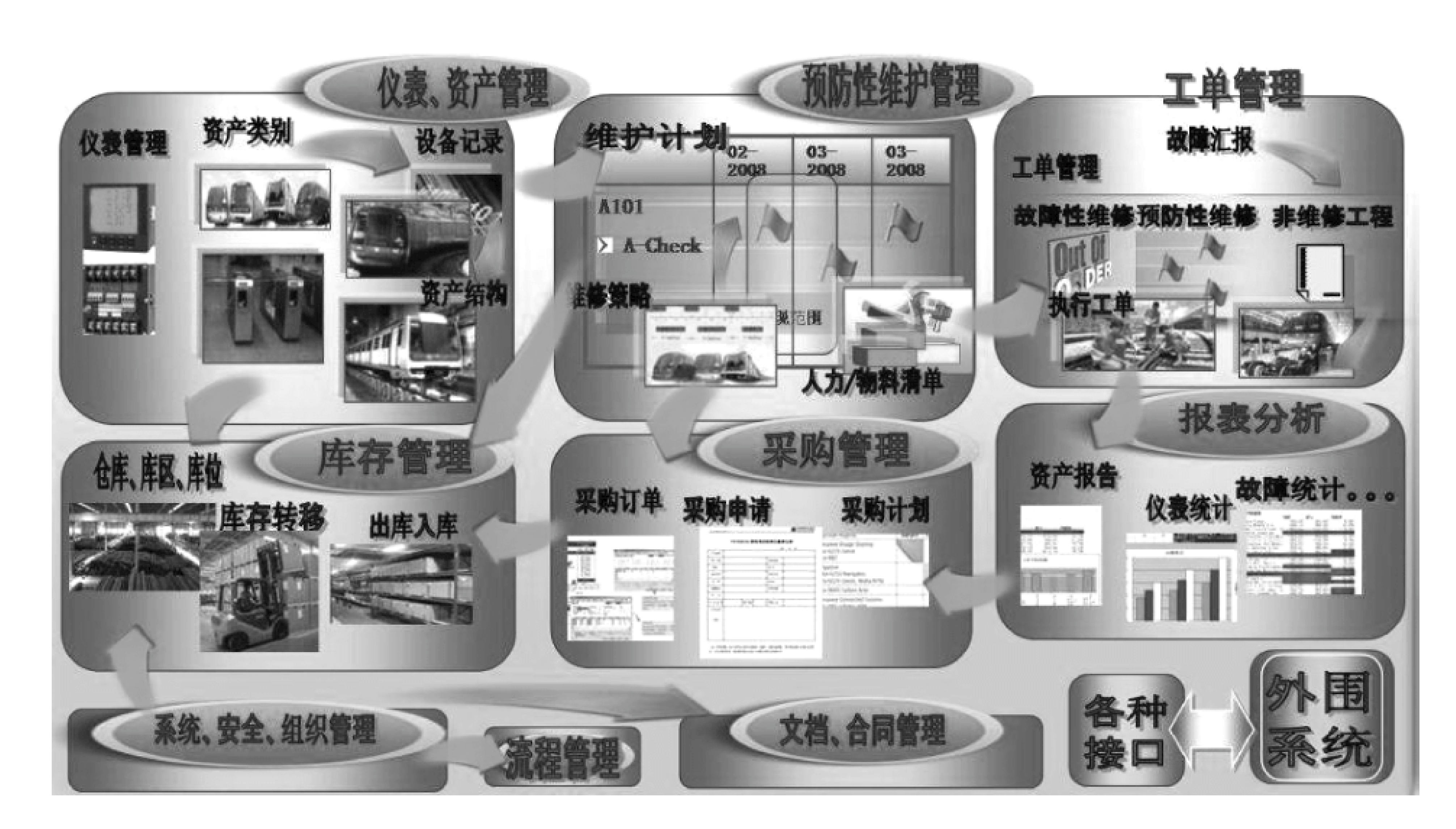


图 9-46 系统主要功能框架

生产力,提高库存效率,增强执行保修合同的能力,为列车服务企业节省时间和金钱。通过MMIS,用户能:

(1) 注册、跟踪并分析整个生命周期内的企业资产。

(2) 为企业资产的周期性维护,灵活建立大量计划。

(3) 运用工单模组,监控各种维护工作。

(4) 通过获取工单上准确的劳务成本及物料支出,分析、优化维护工作成本。

(5) 通过维修计划,预测采购需求。

(6) 通过供应商询价管理,控制采购成本。

(7) 通过库存管理,提高库存效率,降低库存成本。

MMIS 在高阶上划分成 13 个模块。它们分别为设备管理、维护管理、工单管理、采购管理、库存管理、报表分析、文档管理、合同管理、流程管理、安全管理、组织管理、系统管理及个人门户管理,如图 9-47 所示。

1) 设备管理

设备管理模块用于维护与设备相关的基础信息,例如设备登记、设备类别、位置、设备架构、设备结构定义以及设备安装/拆卸管理等。

设备管理模块所有业务数据除了提供手工录入,也支持批量上传及下载。

设备管理模块包括运营机电设备的维护管理,也包括车辆设备的维护管理。

仪表管理提供创建用于设备计划检修或状态检修的仪表、记录设备运行数据、状态数据的功能,从而为设备的计划检修或状态检修提供依据。

设备管理模块提供检修器具台账的创建与管理功能,支持对检修器具进行定期检修计划,以及员工器具配置、领用、归还等管理功能。

2) 维护管理

维护管理包含预防性维修、故障性维修和状态性维修 3 类,用户可根据不同类计划的特点、现实业务需求,对不同设备类型进行科学、合理的计划编排。同时,系统能根据时间自动

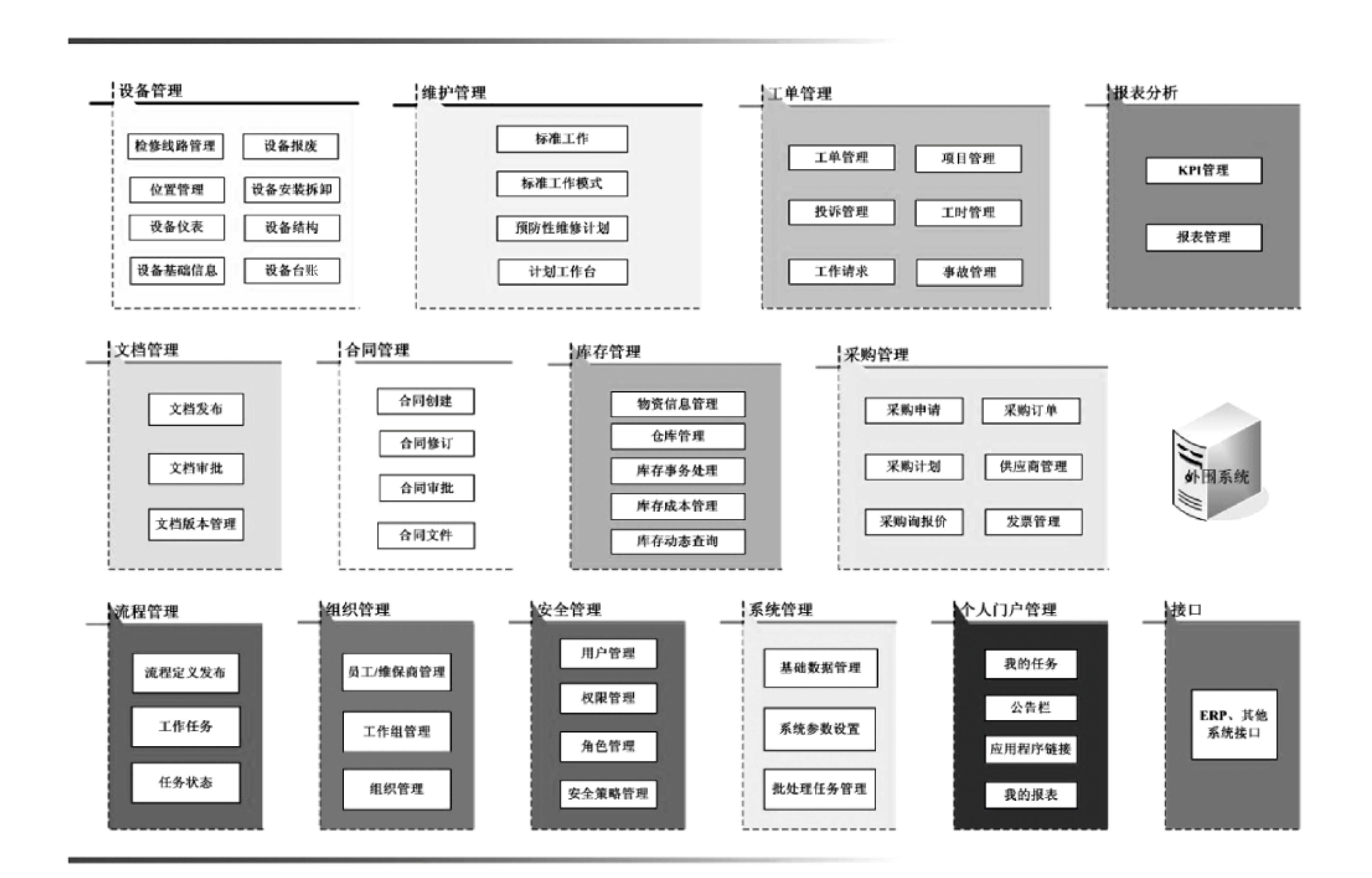


图 9-47　系统模块结构图

提醒计划编制人员即将到期的计划工作，还可支持手动和自动创建模板，然后用户填写数据，自动上传到系统。

预防性维修管理是企业实行设备预防维修，保持设备状态经常完好的具体实施计划，其目的是保证企业生产计划的顺利完成。

预防性维修管理工作主要包括根据当前产品及新产品对设备的技术要求和设备技术劣化程度，编制设备维修计划并认真组织实施。在保证维修质量的前提下，完成维修计划，缩短停修时间和降低维修费用。

3）工单管理

工单管理是通过工单，解决计划性的、非计划性的维修工作，把复杂、庞大的维修工作细分，有阶段、有计划地逐步完成。

工单管理包含工作内容、工作要求，还包含资金预算信息、需要的物料信息、相关的项目信息，是工作流、资金流和物流结合在一起的载体，也是将采购和库存连接起来的一个桥梁。系统运行采用多种标准生产工单，包括基于时间频率和基于度量的，还可以根据站点、测量点等生成工单。

工单管理包括工单、工作请求、工时表和事件。

4）采购管理

采购管理处于资产生命周期的获取阶段，主要包括设备的采购申请、采购订单管理、采购计划管理、供应商及询价管理、发票管理等。

好的采购要做到 5 个恰当：恰当的数量、恰当的时间、恰当的地点、恰当的价格和恰当的来源。采购策略的正确选择是做好采购工作的前提。

5）库存管理

库存管理模块主要包括物资信息管理、仓库管理、库存事务处理、库存成本管理及库存动态查询等。

6）合同管理

合同管理模块提供对企业运营和维护活动相关的合同进行维护管理的功能，如合同分类、合同创建、合同审批、合同修订、合同审批等。

合同管理对计划和建设部门、财务部门也是相当重要的一环；可实现合同信息被相关部门共享、查阅、存档。

7）文档管理

文档管理可以让用户多路同时存取、更加安全的访问控制、更容易搜索与查阅、审计记录查询、有效的版本控制、没有实际空间存储问题和损毁问题、文件保持与跟进措施的提醒、向无纸化办公发展。

文档管理模块支持多种格式的文档管理；支持文档流程化审批，流程可以依据企业业务和组织的变化而进行修改；更严密的安全访问控制有益于预防未经授权的访问；管理文档的不同授权级别；支持文档的版本控制，文档版本的控制能保证文档的准确性、唯一性和现时有效性；支持文档的关联，能将文档与设备、工单、项目等关联，利于企业的信息共享。

8）报表分析

报表管理能够随时让用户全面、清晰、直观地了解企业业务总体及细节运作状况。其良好的报表管理能显著提高报表传输和共享的效率，并且降低报表的相关费用，低成本地为公司领导的决策进行有力的支持。

9）流程管理

流程管理模块为用户提供定制工作流、查看系统分派任务的功能。

利用流程管理模块提供的流程设计器，可以针对工单流程、工作请求流程、文档审批流程等进行客户化定制。

10）安全管理

系统提供对用户账户、用户角色和访问权限等功能的管理。每个用户都具有独特的身份进入系统。这些功能对于系统的安全性是必不可少的，特别是对认证和授权。系统利用用户账号实行认证，利用角色和访问组、字段权限组等实行授权。

11）组织管理

组织是工作组和个人的群组，它拥有多层的架构。例如，可能会有 3 层的组织：公司、部门及室。室从属于部门，而部门则从属于公司。不同层的组织之间存在着层级关系。各个组织可以有数个工作组。设备的数据访问权限、工单或其他相关数据将会指定子组织。权限将会在组织的不同层级上生效。

12）系统管理

系统管理模块提供系统参数配置、会计期间、账号段、工作类型、物料信息等基础数据维护功能，主要分为基础数据管理、系统参数设置和批处理任务管理。

13）个人门户管理

支持个性化的设置，针对每个用户，提供其个人使用的工作环境和访问界面，用户可以依照个人日常操作的需要和习惯，选择相关栏目，设计个人的版面布局，将门户定制成个人

的工作平台,进行工作进度跟踪、查看报表、KPI 指标、设备状态等工作。

14) 与系统的接口

PCCW MMIS 系统主要与财务系统、HR 系统等进行信息交互。例如,在津澳地铁 MMIS 项目中,其发送与接收的接口信息如表 9-3 所示。

表 9-3　PCCW MMIS 系统接口表

源系统	目 标 系 统	数 据 内 容
GL	本系统	账号段
FCS	本系统	F1 Number
INV	本系统	物料代码、单价、可用数量
HR	本系统	员工信息
INV	本系统	工单的实际物料成本
AP	本系统	工单的实际其他成本
本系统	GL	工单的实际人力成本
本系统	INV, PO, AP	工单号

3. 产品优势

(1) 电讯盈科既是 MMIS 产品的开发商,又是实施服务的提供商。

产品开发商与实施商为同一家公司的益处,有以下几点。

① 降低成本;

② 增加效率;

③ 提高实施质量;

④ 更能贴近用户需求,客户化定制程度较高;

⑤ 降低实施风险,提高成功率;

⑥ License 限制比较宽松。

(2) MMIS 产品具有以下独有的价值优势。

① 符合国际级的资产管理标准(PAS-55);

② 融合优秀地铁公司多年行之有效、久经考验的业务流程和成功的资产管理经验;

③ 结合资深顾问服务,面向铁路运营单位,针对轨道交通业务为核心的优化方案;

④ 满足铁路 RAMS 的要求;

⑤ 对轨道等连续性设备的有效管理;

⑥ 能和其他系统无缝集成;

⑦ 简单易用、直观友好的用户界面,支持多语言;

⑧ 先进的信息技术、业界标准和开放的平台,有足够的灵活性及可扩展性;

⑨ 易于实施,降低实施风险和成本。

4. 实施方法

电讯盈科在中国铁路实施信息系统及相关业务系统已有多年,对资产核算、全生命周期资产管理等业务有较深理解。在本项目之前,电讯盈科已在部分铁路公司兄弟省份建成了与本项目需求极为类似的业务模块/系统。因此,对于本项目的建设,我方将根据铁路关于

资产全生命周期管理实施项目的标书要求，结合此前的业务咨询和系统建设规划成果，充分利用电讯盈科为其他铁路公司的建设成果，进行有机整合，迅速、准确地开发出满足要求的系统，并完成项目实施任务。进行项目实施所遵循的方法论如图 9-48 所示。

电讯盈科快速实现了全球公认最佳作业的港铁资产维修流程和方案，使其拥有了一整套功能强大并符合地铁行业特点的系统，提高了业务效益；快速实施低成本低风险的世界级的系统，更避免了实施 ERP 软件所需的昂贵客户化定制和咨询顾问服务费用；用最低的总成本，使客户拥有一套由经验丰富的地铁资产管理专家设计、直观易用、开放技术平台的资产管理系统。

系统实施涉及的阶段如下：项目定义阶段、项目设计阶段、项目构架阶段、项目转换阶段及项目运行阶段。每一个轮次的 CRP 或者 UAT，均需经历如下环节：案例准备或调整、环境准备、数据准备、数据导入、测试、问题日志回顾、问题处理、软件发布、总结等。

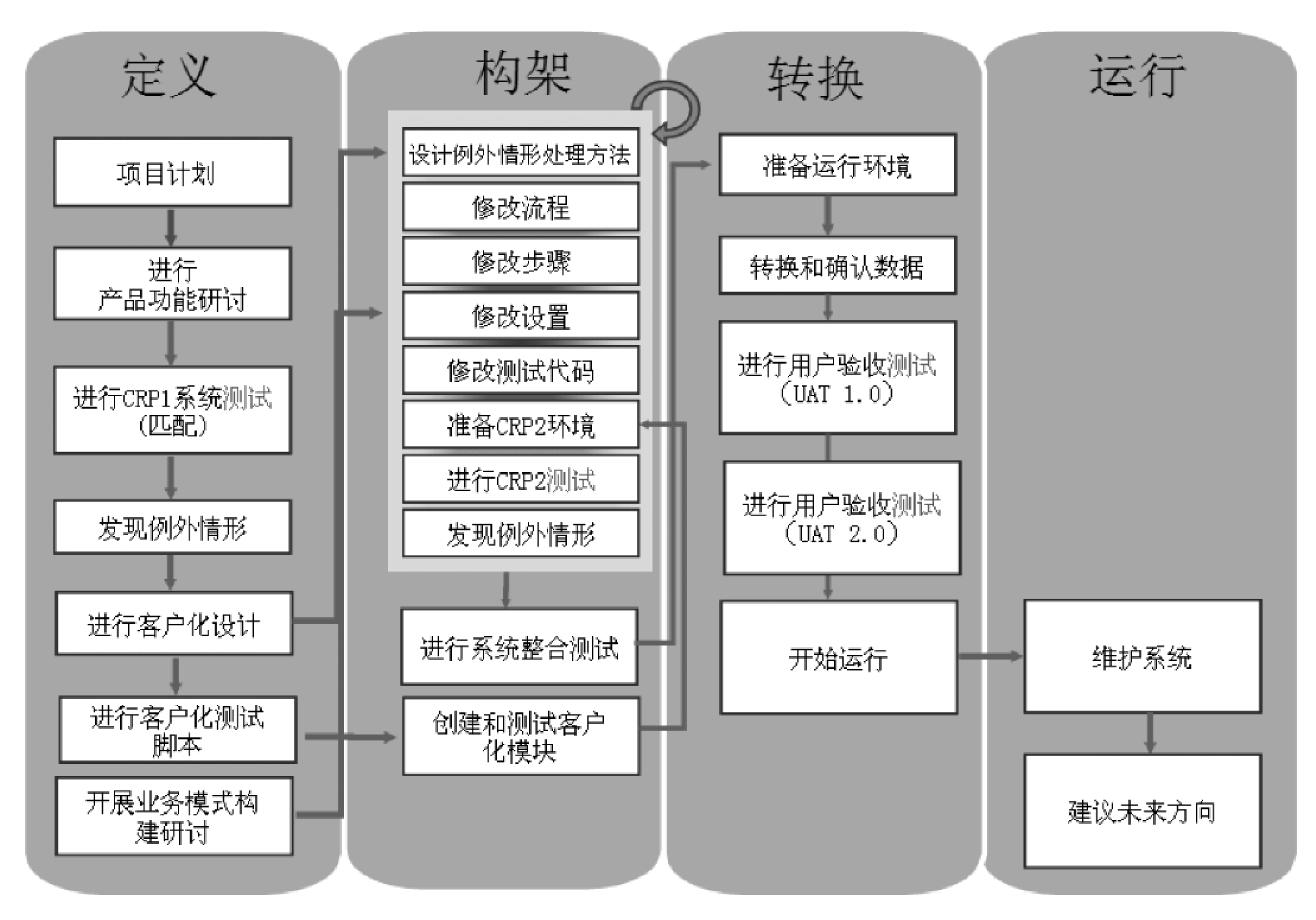


图 9-48 项目实施方法论

注：CRP 即会议室导航，用于帮助项目组建立业务流程模型。通过 CRP 还可发现软件与现实的差异，并进一步调整软件配置。

电讯盈科对项目所涉及的管理与控制包括项目计划、项目状态报告(周报、月报)、问题管理、变更管理、配置管理、质量管理、风险管理、关联管理、项目跟踪及监控。

项目组成员在项目过程中，将所有问题记录在问题日志中，由电讯盈科项目经理或其代理统一记录、更新和跟进。

问题日志包括如下内容：问题提出时间、问题描述、优先级、问题提出人、问题处理负责人、问题状态(如：未解决、正在解决、已关闭、已取消)。当一个问题解决后，相关的项目组成员要通知电讯盈科项目经理，更新相应的问题状态和必要的信息、问题的解决方法、问题

的解决时间。

对于需要变更的内容，需对其进行控制，履行必要的复核手续，并加以标示和报告。如果关系到项目成本和关键里程碑的变动，需要得到项目指导委员会的批准。

系统上线前，电讯盈科负责配置库清单的更新，配置库清单的格式按照甲方客户的信息系统项目管理方法。上线后，将配置库清单提交给甲方客户。

通过对项目的风险进行识别、分析，并采取必要的纠正或预防措施来降低风险。

北京津澳地铁有限公司是一个致力于地铁建设、运营、管理的专业化公司。公司是在北京市加大基础设施投资体制改革力度的历史条件下，由北京市基础设施投资有限公司、北京首都创业集团有限公司和中国香港铁路有限公司共同出资组建。该公司采用的 EAM 系统即是电讯盈科提供的 MMIS 产品，主要用于对车辆及设备进行科学管理，对维护维修活动做出合理安排，以提高设备有效使用率。系统主要的功能包括设备管理、维护计划管理、工作管理、预算和报备、系统管理等。

2008 年 7 月，电讯盈科承接津澳地铁有限公司的设备维护管理信息系统（MMIS）的开发与实施项目。根据合同要求，需要在上线前对该系统进行压力测试。

9.3.2　RPT 和 LR 的对比分析

首先，进行 MMIS 系统压力测试面临的工作是压力测试工具的选型问题，目前市面上比较流行的两种性能/压力测试工具是 IBM Rational Performance Tester（简称 RPT）和 HP Mercury LoadRunner（简称 LR）。

我们知道，LoadRunner 是一种适用于各种体系架构的自动负载测试工具，通过模拟实际用户的操作行为和实施实时性能监测，来帮助用户排查和发现问题。相比于 RPT，LR 能支持更广泛的协议和技术，适应面很广，为用户的特殊环境提供特殊的解决方案。LR 的组件很多，其中最核心的组件包括以下几个。

（1）Vuser Generator（VuGen）用于捕获最终用户业务流程和创建自动性能测试脚本。

（2）Controller 用于组织、驱动、管理和监控负载测试。

（3）Load Generator（负载生成器）用于通过运行虚拟用户生成负载。

（4）Analysis 用于查看、分析和比较性能结果。

RPT 也是一款性能测试工具，适用于基于 Web 的应用程序的性能和可靠性测试，将易用性与深入分析功能相结合，从而简化了测试创建、负载生成和数据收集，以帮助确保应用程序具有支持数以千计并发用户并稳定运行的性能。

（1）RPT 是针对 Web 应用程序的性能测试工具，基于 Windows 和 Linux 的用户界面，使用基于树状结构的测试编辑器提供高级且详细的测试视图。

（2）提供不同用户数的灵活的模拟，支持将 Windows 和 Linux 用作分布式负载生成器，使用最小化的硬件资源实现大型、多用户的测试。

（3）支持使用自定义 Java 代码的灵活测试定制。

下面主要从脚本开发、场景构建与配置、性能监控、测试结果分析 4 个方面对 RPT 和 LR 这两个性能/压力测试工具的使用进行详细的对比分析，并根据实践经验总结 RPT 的一些实用技巧。更多的了解和学习可以访问其相关网站。

1. 脚本开发对比

LR/RPT 的脚本的开发过程通常都是采用录制＋定制的模式。首先通过对典型业务逻辑的录制，完成脚本中的基本业务的框架，然后针对录制结果，通过参数化、数据关联、增加逻辑控制等方式加强脚本的适应性来满足特殊的业务需求。

1) 脚本录制/定制过程

(1) LR：直接生成面向过程的运行代码。

LR 通过对基本业务的录制，VuGen 将生成 Vuser 函数(也称作 LR API)，并将它们插入到脚本中。在实践中，LR 脚本就是由这样的 Vuser 函数和一些定制代码组成的。对于基于 Web(HTTP/HTML)的应用程序的测试，多数用户选择基于 C 语言的 LR 脚本，显然，这种 LR 脚本是一种面向过程的脚本，开发者可以对最终运行的脚本进行直接的修改与调整。对于开发者来说，这种 LR 脚本的开发方式比较灵活。相应地，这项工作，对于开发者的编程基础，尤其是 C 语言和 LR API 的了解，要求都比较高。

(2) RPT：录制结果经过“翻译”生成最终的运行代码。

与 LR 不同，RPT 的脚本录制过程可以拆分成两步。如图 9-49 所示，第一步，RPT Recorder on RAC 负责记录用户的所有 HTTP 请求，生成一系列的 Trace 文件。Trace 文件记录了用户与服务器的交互过程。第二步，当用户完成脚本的录制过程之后，RPT Test Generator 能够根据 Trace 文件“翻译”一遍，生成最终运行的测试脚本。

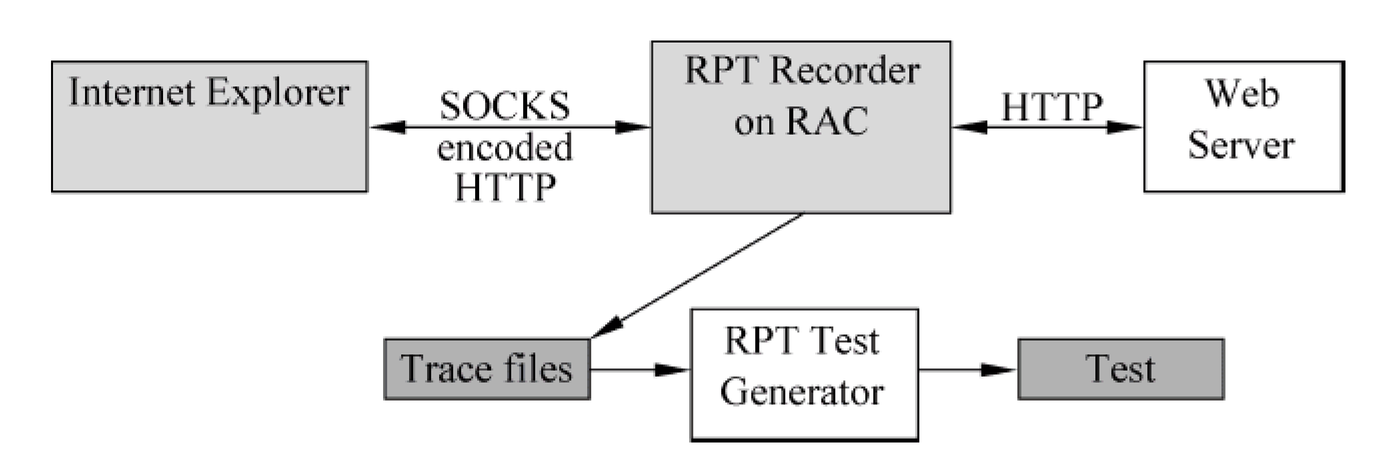


图 9-49 RPT 脚本的录制和生成架构

这种生成临时 Trace 文件的好处是用户可以随时依据该 Trace 文件生成新的测试脚本，然后再对脚本进行测试场景定制，而不用对同一个操作过程做多次录制操作。

2) 参数化

录制业务流程时，LR/RPT 生成一个包含录制期间用到的实际值的脚本。假设用户要使用不同于录制内容的值执行该脚本的操作时，就需要用参数替换已录制的值，这被称为脚本参数化。脚本的参数化可以简化脚本，同时增强脚本适用性。对于 LR 和 RPT 脚本，参数化过程类似，都是定义参数，为参数指定属性或者数据源的过程。但是在 LR 中，只有函数中的参数才能参数化，除此之外，其他字符串不能进行参数化。

RPT 的参数化过程同样简单(以替换用户登录密码为例来说明)，首先，选中需要进行参数替换的请求页面，如图 9-50 所示，选中左侧的登录请求页面。在其右侧的 Test Data 中显示与该请求页面相关的所有数据信息，脚本录制人员可以用其他值代替图 9-50 中的 password 变量。

3) 数据关联

数据关联类似于参数化，可以简化脚本，适应企业应用中需要动态数据的情况。默认情

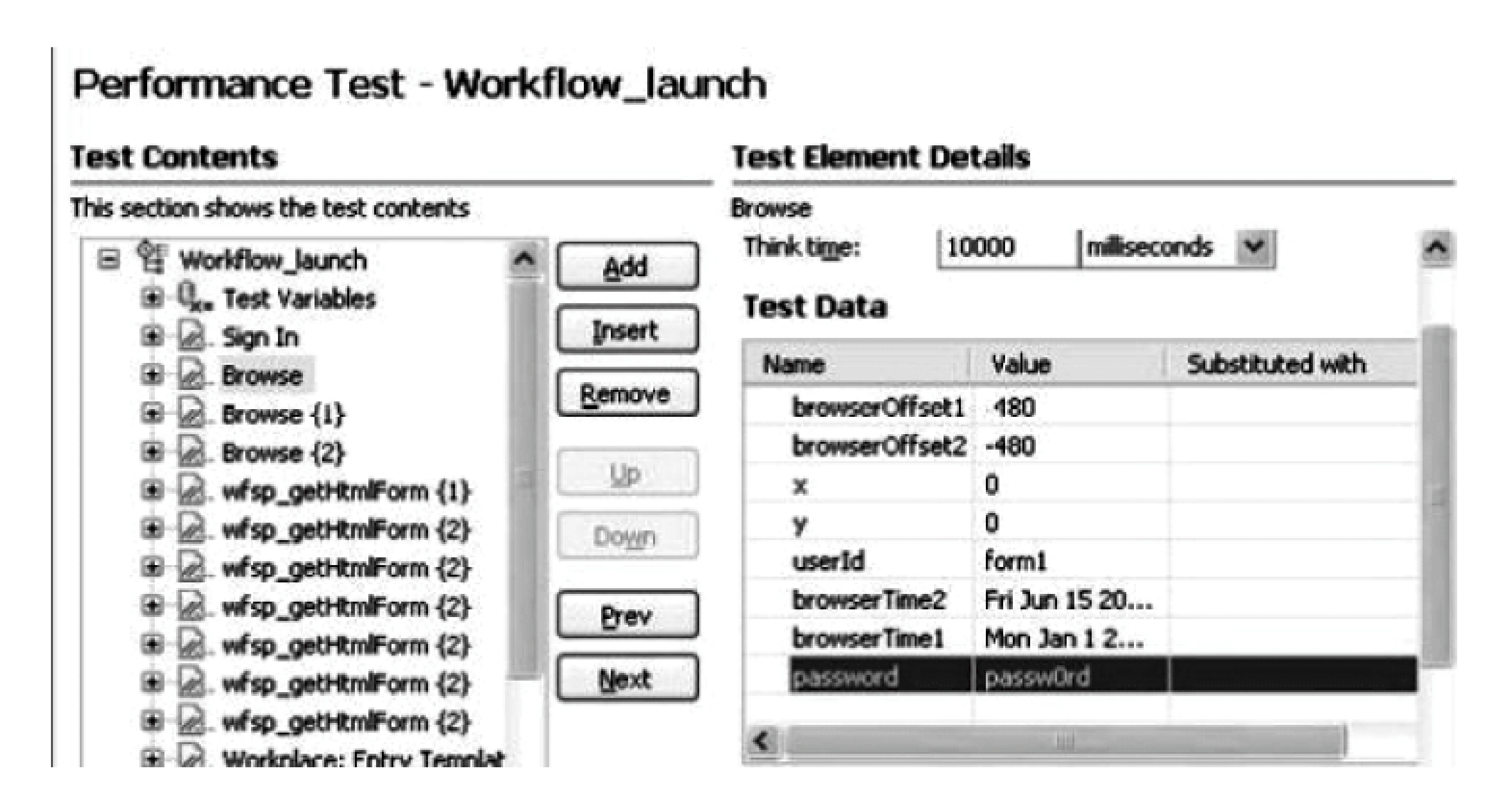


图 9-50　RPT 脚本参数化

况下，LR 和 RPT 都能做到一些基本的数据关联，但是由于 HTTP 请求之间关联的复杂性，需要用户手动做一些数据关联。数据关联包含 3 个步骤，一是定义哪个录制的值需要被关联（替换）；二是定义数据源；三是定义被关联的数据与数据源之间的关联关系。

LR 的数据关联过程如下：lr_save_XXXX(value,dataSource)语句将数据源的值保存到参数 dataSource 中；用 lr_eval_XXXX(dataSource)语句替换被关联的数据。

RPT 中如果需要自己定义关联，则在 HTTP 请求中的 URL 中或者 Data 中选择需要创建关联的部分，然后右击选择替换对象。其中，替换对象可以是脚本中已经建立的引用（这里的引用就是一种用户自定义的数据源），或者 RPT 自带的数据源（例如时间戳对象），或者是 Custom Code。

图 9-51 中浅色的部分是已经被关联的 URL，运行测试时该部分将由被引用的 URL 值来替换。

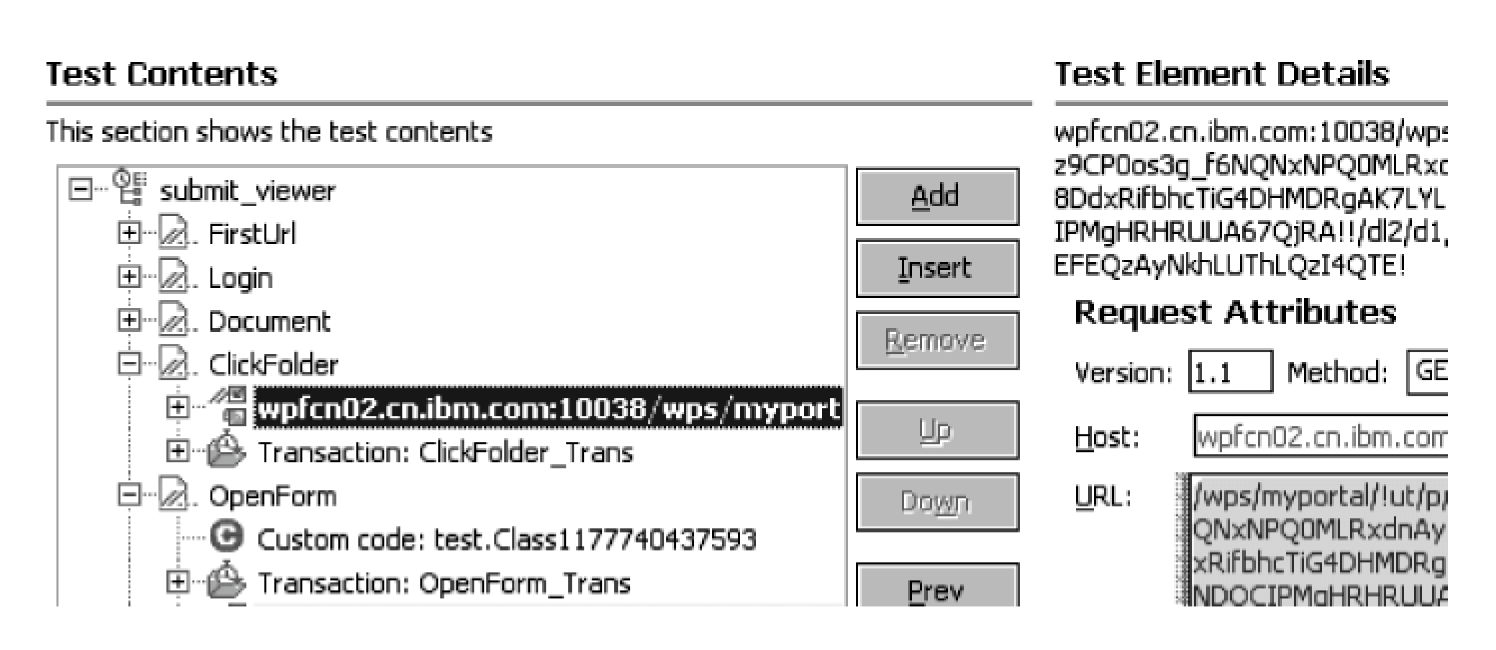


图 9-51　RPT 数据关联

4）Custom Code

Custom Code 是 RPT 独有的概念。尽管 RPT 脚本开发过程中，用户可以直接在 UI 层面达到对脚本的定制，但是这种定制能力毕竟有限。将定义好的 Custom Code 通过 UI 穿插到脚本中，为 RPT 录制的脚本提供充足的扩展能力来保证其灵活的定制性。Custom Code 本质上就是一个 Java 类。Custom Code 需要实现 com.ibm.rational.test.lt.kernel.custom.ICustomCode2 接口，并实现该接口中的如下方法：

```
public String exec(ITestExecutionServices tes, String[] args)
```

ITestExecutionServices tes 参数是 Test Container 中的一个实例，使用它可以访问 Test Container 运行态的一些服务。

String[] args 参数是定义 Custom Code 时传入的参数数组。

该方法的主体就是基于传入的信息进行业务逻辑处理代码，然后将处理结果（一个字符串）返回，其返回的字符串可以被后续的请求引用。Custom Code 是一个纯 Java 的类，所以具有 Java 编程经验的人都可以根据业务需求编写自己的 Custom Code。

5）数据池

性能/压力测试过程中，通常都需要为某些测试提供大量的测试数据。LR 和 RPT 都提供了数据池功能，即将一个数据文件作为参数值赋给一个参数，如图 9-52 所示。

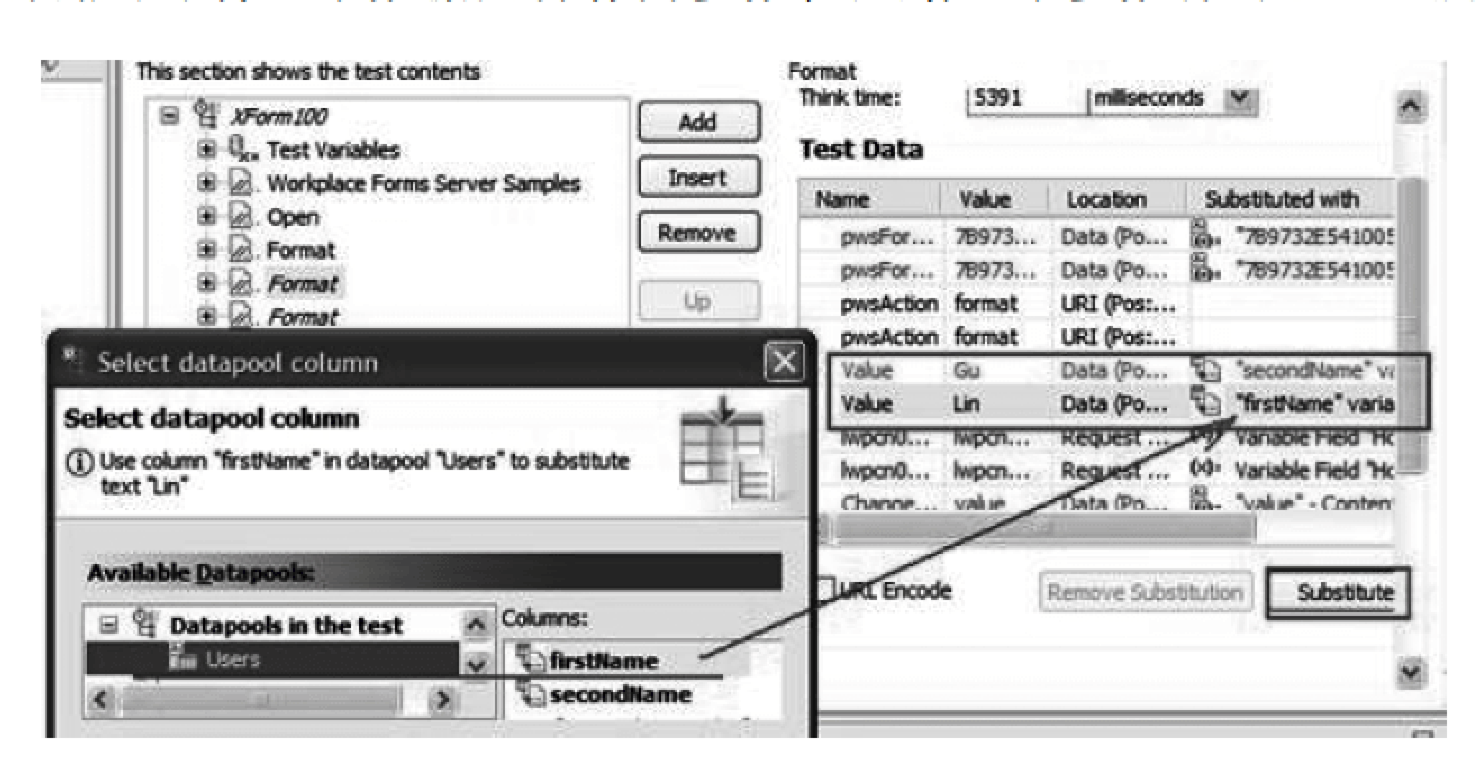


图 9-52 RPT 数据池

LR 中，用户可以指定该文件的多个数据以何种方式赋值到该参数中。LR 提供 3 种选择：顺序，随机，唯一。前两种比较容易理解，最后一种是指每个虚拟用户都从该文件中取不同的值作为参数值，如果该数据池不足够大，所有的备选值都已经被取出过一次，即该数据池资源被用尽时，LR 报错。

在 RPT 中，用户只能顺序从数据池中读入测试数据。RPT 的数据池是以.XML 格式存储的，并且在测试开始时，将数据池中的所有数据都加载到内存中，这样的实现模式不利于测试中使用大数据量。不过灵活的 Custom Code 功能可以弥补这方面的不足。对于大量的测试数据，可以通过自定义 Custom Code 来实现 On Demand 的数据读取和加载。

6）流程控制

LR 脚本大部分是基于 C 语言的，因此 C 语言中的流程控制语句（例如判断、循环等）都可以加入到 LR 脚本中。RPT 的流程控制操作可以通过 UI 界面轻松进行，它提供了灵活的流程控制模式，包括 IF 条件控制结构和 LOOP 循环结构。

（1）iF 条件控制结构

在 RPT 脚本中，可以将一部分连续的页面或者 HTTP 请求放到一个 iF 条件中，然后由判断条件来确定 iF 结构中的页面或者 HTTP 请求是否被执行。其判断条件可以是 RPT 自动参数化后的参数，也可以是 Custom Code 的返回值，或者是数字、字符串等。

图 9-53 是添加了 iF 条件后的脚本，包含为 iF 语句设置判断条件的配置界面。

同样，RPT 也支持 iF-ELSE 的结构。

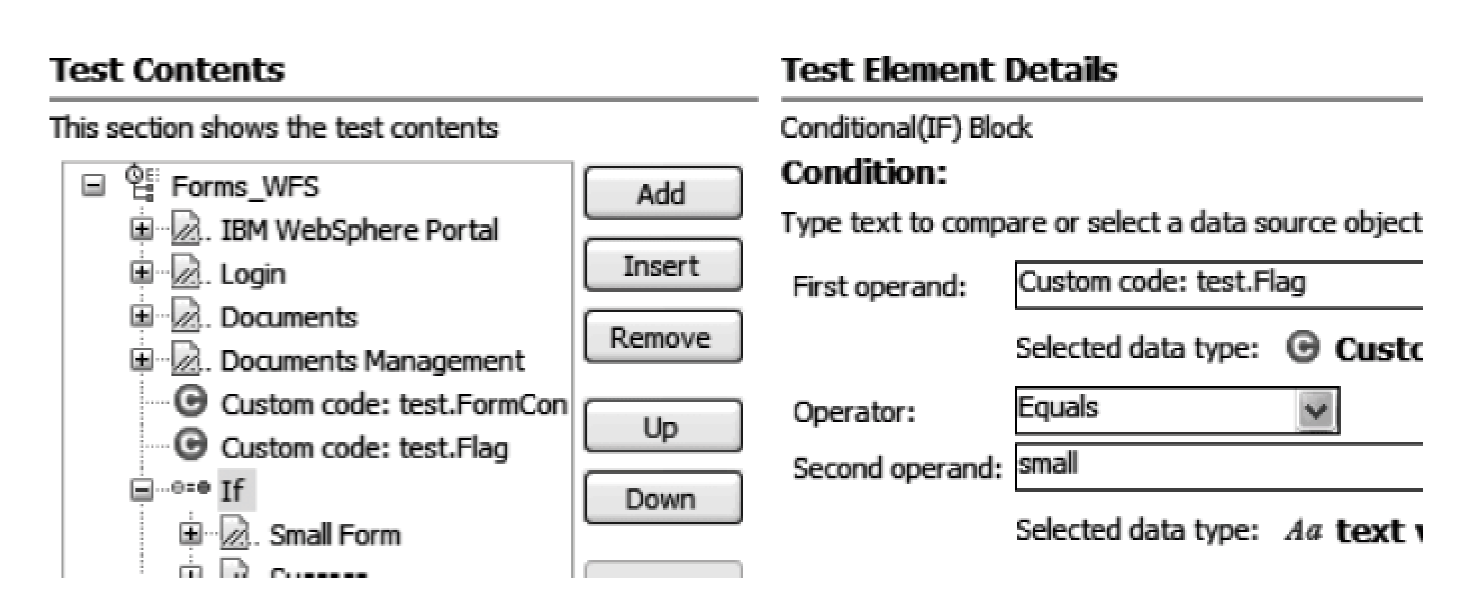


图 9-53　RPT iF 条件控制结构

（2）LOOP 循环结构

RPT 中的另外一种流程控制结构就是 LOOP 结构，如图 9-54 所示脚本中，将所有的页面放到了一个 LOOP 结构中，然后可以通过指定循环次数来确定其中的脚本循环执行多少次。

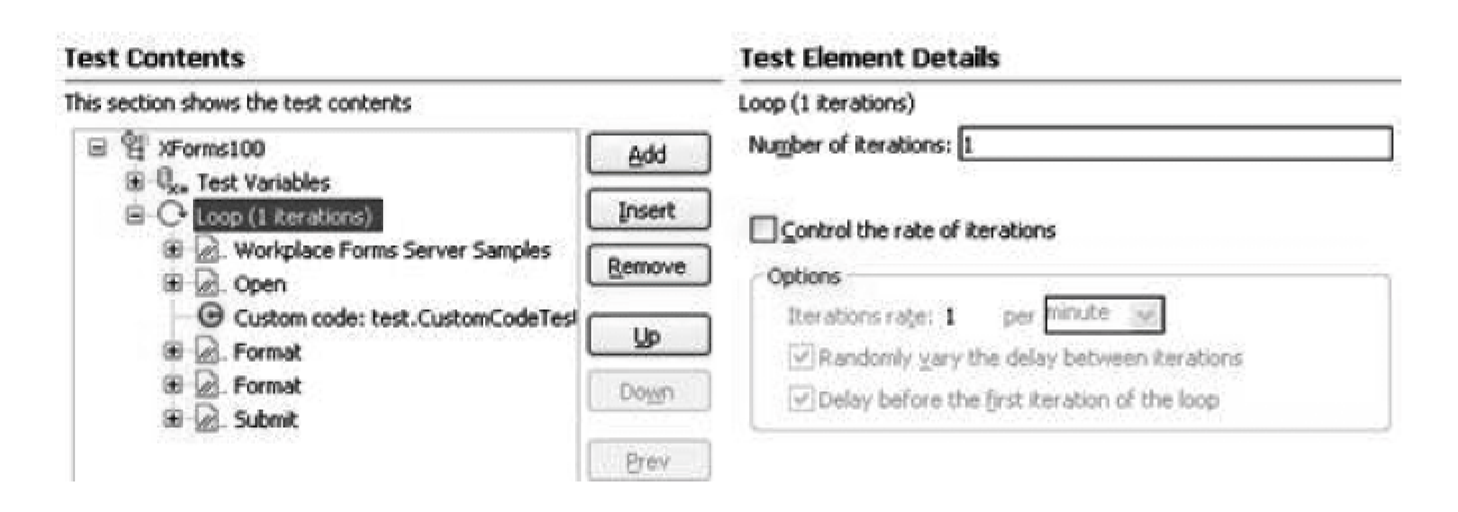


图 9-54　LOOP 控制结构

7）全局信息

这里所说的全局信息，实际上是脚本运行时，LR/RPT 产生的内部数据。例如，当时的运行时间，Vu 所在用户组组名，迭代编号等。在 LR 中，所有的全局信息作为特殊的参数类型，供脚本开发者使用。例如，如果一个变量为 Group Name 类型，则该变量的值即为当前用户所在的用户组的组名。

RPT 中的全局信息，存在 IDataArea 对象中。IDataArea 对象包含 3 个方面的信息，分别是 Test Data、Virtual User Data、Engine Data，这些信息都可以通过 Custom Code 来获得。

Custom Code 的实现需要继承 ICustomCode2 类，并实现该接口的核心方法 public String exec(ITestExecutionServices tes, String[] args)，该方法的第一个参数就可以获得 IDataArea 对象，然后获得全局信息。同时用户也可以向 IDataArea 对象中添加信息，提供给测试脚本的其他地方使用。

8）错误控制

LR 用户可以指定脚本执行期间的错误的处理方法。默认情况下，当脚本执行出现错误，脚本将退出执行。用户也可以配置运行设置指示当 Vuser 出现错误时仍继续执行脚本。除此之外，用户还可以在脚本中加入 lr_error_message 函数，便于对日志的分析。

RPT 中，当脚本运行出现错误，脚本将继续执行，可能后续会出现很多错误。

9）Debug

LR VuGen 可用作常规文本编辑器。可以在其中打开任何文本文件并进行编辑。当重播期间在输出窗口中显示错误消息时，可以双击该错误消息，VuGen 将使光标跳到导致问题的测试行。还可以将光标置于错误代码上并按 F1 键，查看该错误代码的联机帮助。

RPT 在运行完一个测试之后，会产生相应的测试日志，如果在测试过程中发生任何错误，RPT 会以 Message 的形式提示出该请求发生错误。如图 9-55 所示的测试日志中被选中的 Message 表示该 HTTP 请求在引用前面的关联值时发生错误。

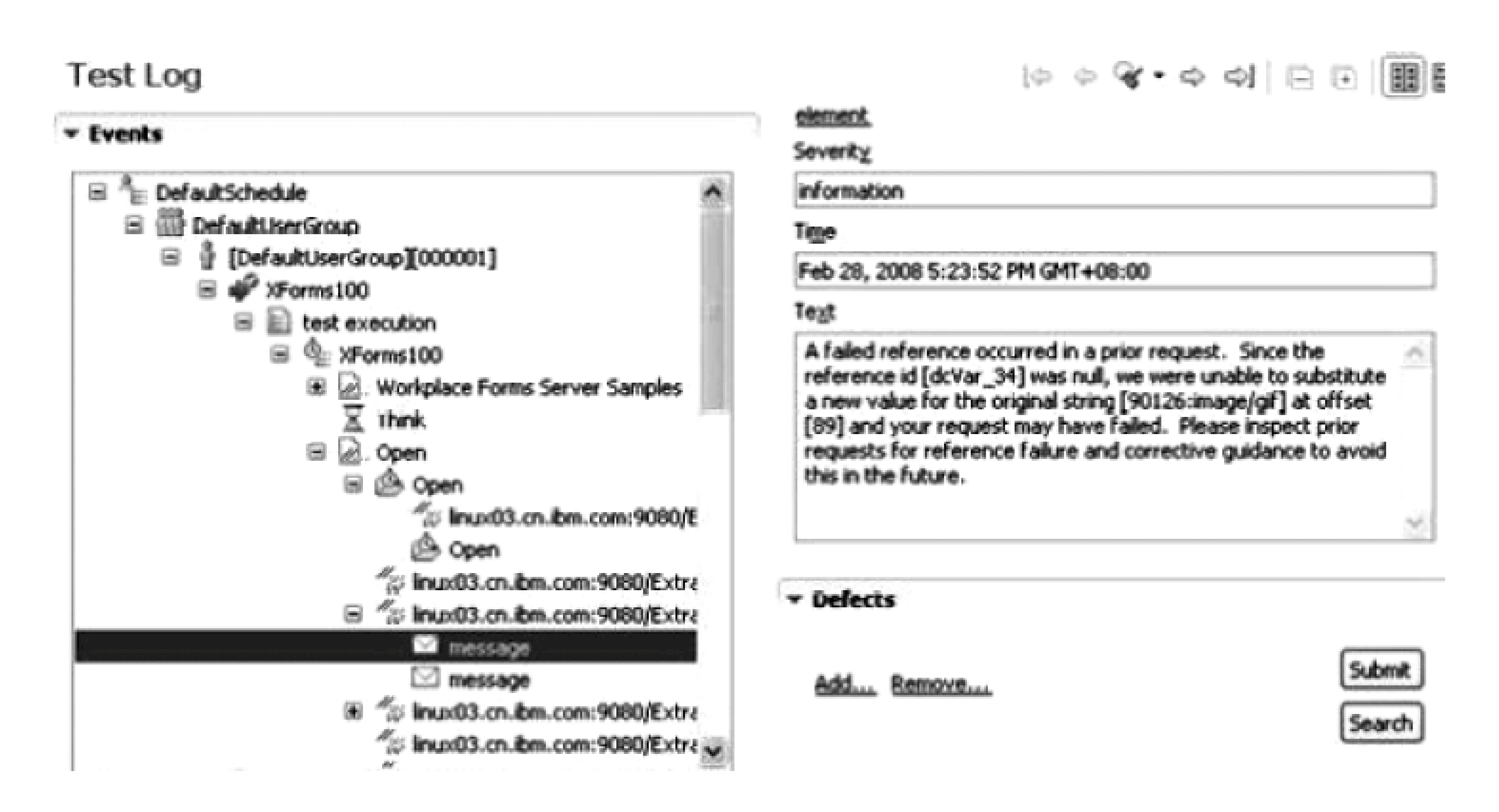


图 9-55　RPT 错误日志

2. 场景构建与配置对比

脚本只是定义了某些用户的操作步骤，而场景则包含有关如何模拟实际用户的所有信息。LR 和 RPT 的场景构建过程比较类似，只是对脚本循环的控制，分配负载生成器等配置上略有差别。

LR 中，Controller 组件负责场景的创建。用户需要在一个场景中添加一个或多个脚本，并为每个脚本分配相应的 Vuser 组。然后，用户可以为每个 Vuser 组分配多个虚拟用户，指定模拟该用户组的负载生成器。图 9-56 中，负载生成器即为本机，即负载生成器跟 Controller 是同一台机器，如果该处配置成其他计算机的 IP 地址，那么该用户组的负载模拟将由其他计算机完成。LR 中不能通过 Controller 指定一个脚本执行与否的概率，但是可以通过 C 语言开发，完成随机调用脚本的功能。Runtime-Setting 包含所有针对该场景的一些附加配置，如脚本循环次数、等待时间、网络模拟等。

Scenario Groups

Group Name	Script Path	Quantity	Load Generators
default	D:\newframework\portal6_wpf27\submit_noviewer_xform100	50	localhost
default 18	D:\newframework\portal6_wpf27\longrun4pdm	20	localhost

Start Scenario / Generators... / Vusers... / Add Group... / Remove Group / Run-Time Settings / Details...

图 9-56　LR 场景创建

RPT 中测试场景是通过 Schedule 来组织的。Schedule 通过用户组、循环控制、随机选择器等功能部件来组织测试脚本使其满足实际场景。用户组则由循环控制或者随机选择器，再加上测试脚本等元素组成。循环控制用来控制其下的测试脚本需要循环执行的次数。随机选择器是为了实现在多个测试脚本中随机选择一个来执行。可以为随机选择器指定权重，通过权重值来决定在多个随机选择项中某项被随机选中的概率大小。图 9-57 是一个 RPT 创建场景的例子，其中，Schedule Element Details 提供了对 Schedule 的丰富配置功能，与 LR 中的 Run-Time 配置功能类似。

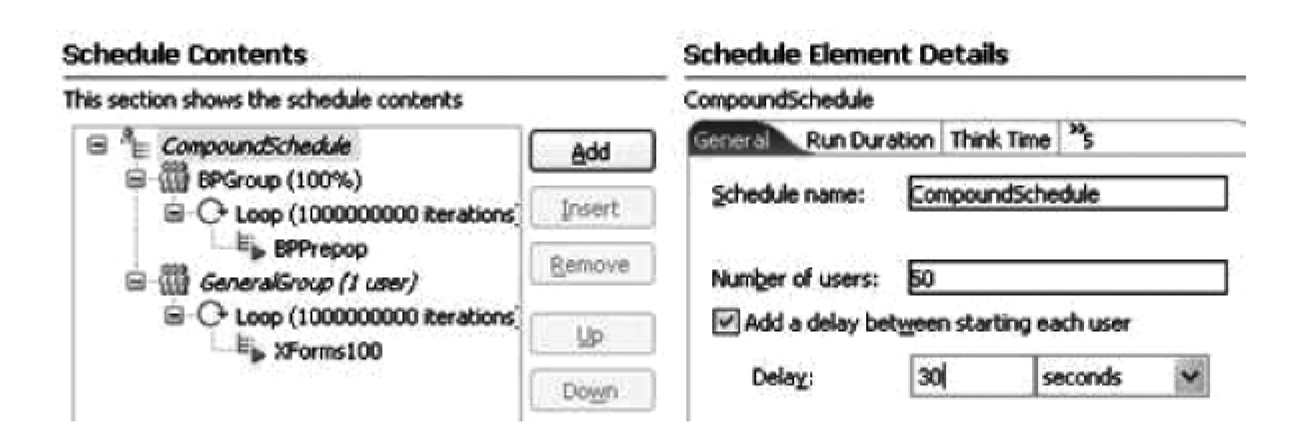


图 9-57　RPT 场景创建

RPT 的负载生成器配置也比较简单，首先选中要放到其他 RPT 主机上进行模拟的 User Group，然后在其配置界面中选中 Run this group on the following locations，在其中添加远端的 RPT 主机信息，如图 9-58 所示。

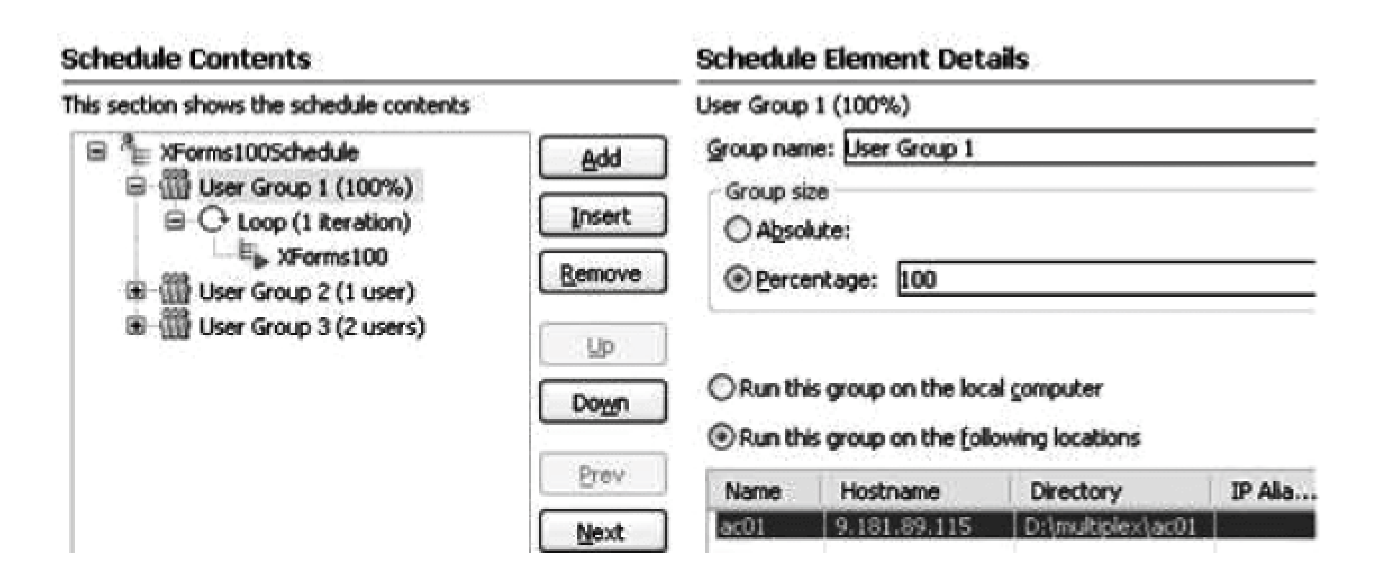


图 9-58　RPT 负载生成器配置

3. 性能监控功能对比

LR 和 RPT 内部都集成了一些实时监控器，RPT 可以对事务、Web、系统、Web 应用服务器等资源进行实时监控。LR 的监控范围更广泛一些，除了上述资源之外，还可以对网络、防火墙、Web 服务器、数据库、ERP、Java 等资源进行实时监控。无论使用哪种测试工具，在自动测试过程中的任何时间，用户都可以获知系统的多种性能指标的当前值和变化趋势。

在一个测试场景中，用户需要将被监控的服务器信息加入到资源监控列表中。LR 中，如图 9-59 所示，从左侧资源树中选择资源种类，在右侧对应资源状态显示窗口中，右击添加被监控的服务器名称。

RPT 中，如图 9-60 所示，用户需要在 Schedule 的配置窗中的 Resource Monitoring 选项卡中添加需监控的服务器。

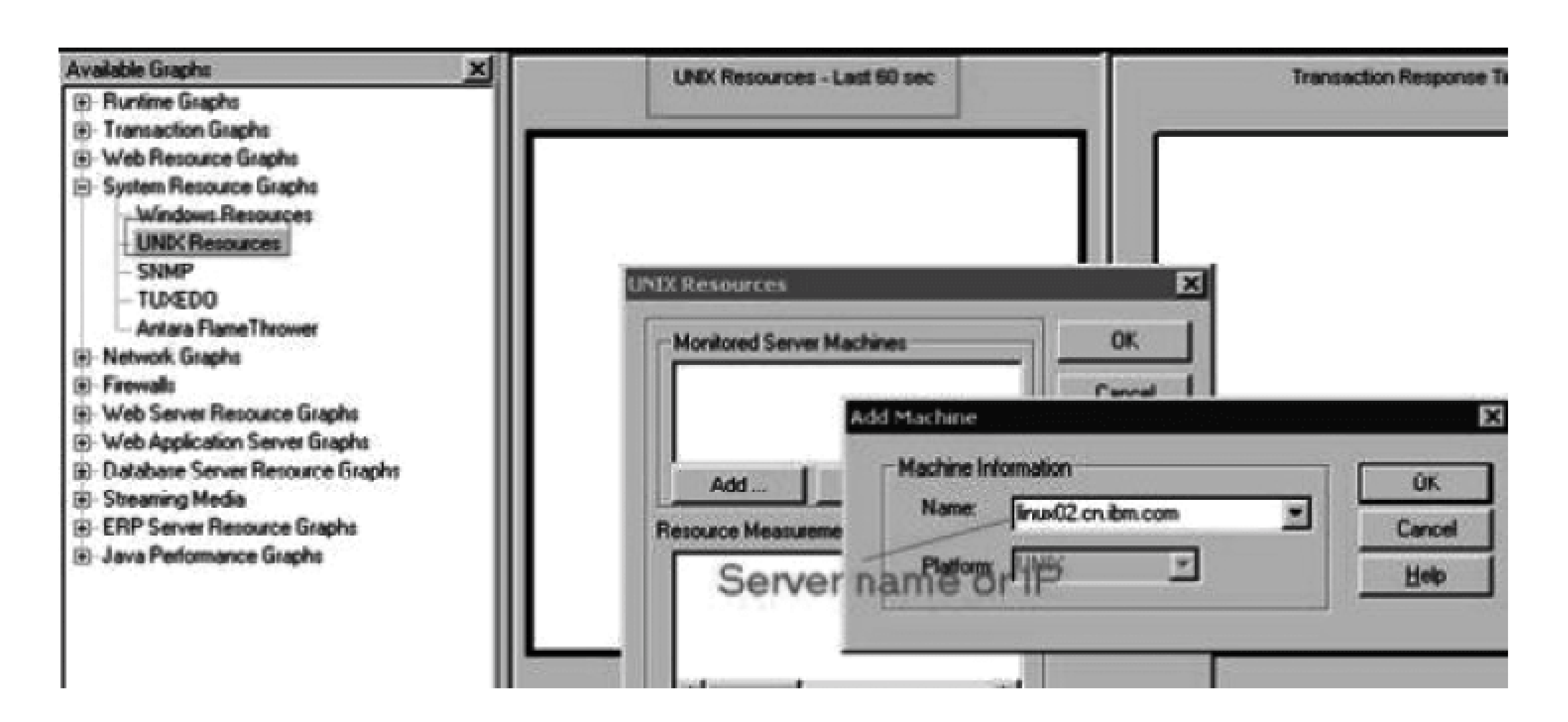


图 9-59 LR 添加被监控的服务器信息

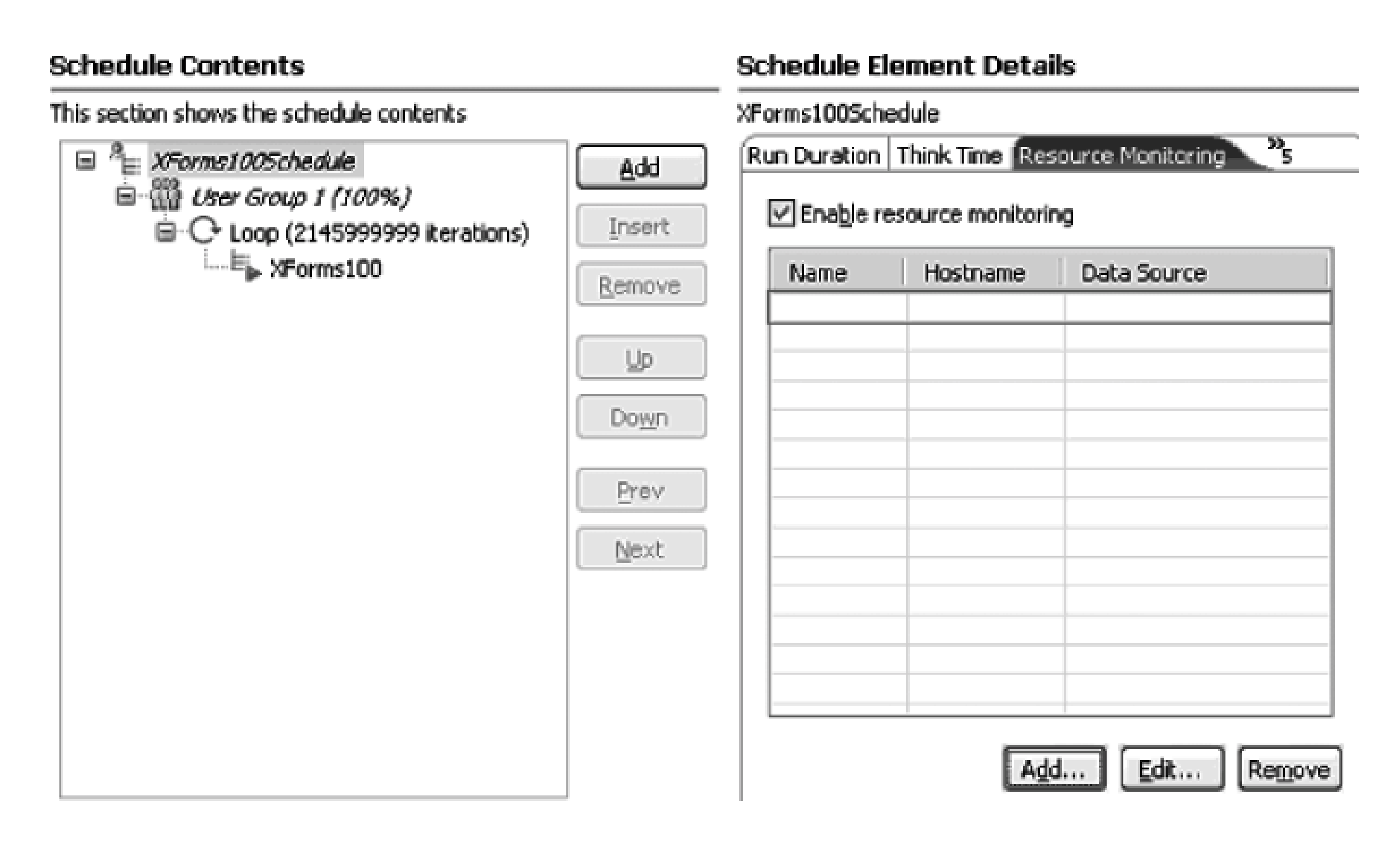


图 9-60 RPT 添加被监控的服务器信息

4. 测试结果分析功能对比

LR 和 RPT 都提供了测试结果的多种图表以及图表之间的叠加效果，方便用户分析测试结果。LR 中测试结果图表的生成由 Analysis 组件完成。默认情况下，用户可以直接看到测试的总体概要分析、吞吐率、事务平均响应时间等图表，如果用户希望看到其他资源的监控图，可以通过添加图表完成(图 9-61)。Merge Graghs 可以帮助用户将同一个测试结果中的多种资源的结果进行叠加(图 9-62)。Cross Result 可以生成多次测试结果的比较分析图(图 9-63)。

RPT 也为测试结果提供了直观的图表表现(图 9-64)，默认情况下，用户可以直接看到对测试成功率的总体柱状图、整个测试过程完成的总体信息列表、测试中页面的反应时间曲线图等报告。也可以通过添加其他监控信息的方法，将其他资源的监控图叠加到当前的监控图中。

RPT 也可以实现多次测试结果的比较。在 RPT 的 Test Navigator 中选择待比较的测试结果，在其右键菜单中选择 Compare，打开 Compare Results 窗口，然后将需要做比较的测试结果添加进来，在下一步中选择要显示的报告，单击 Finish 按钮打开比较结果的显示页面。如图 9-65 所示是一个测试结果的比较报告。

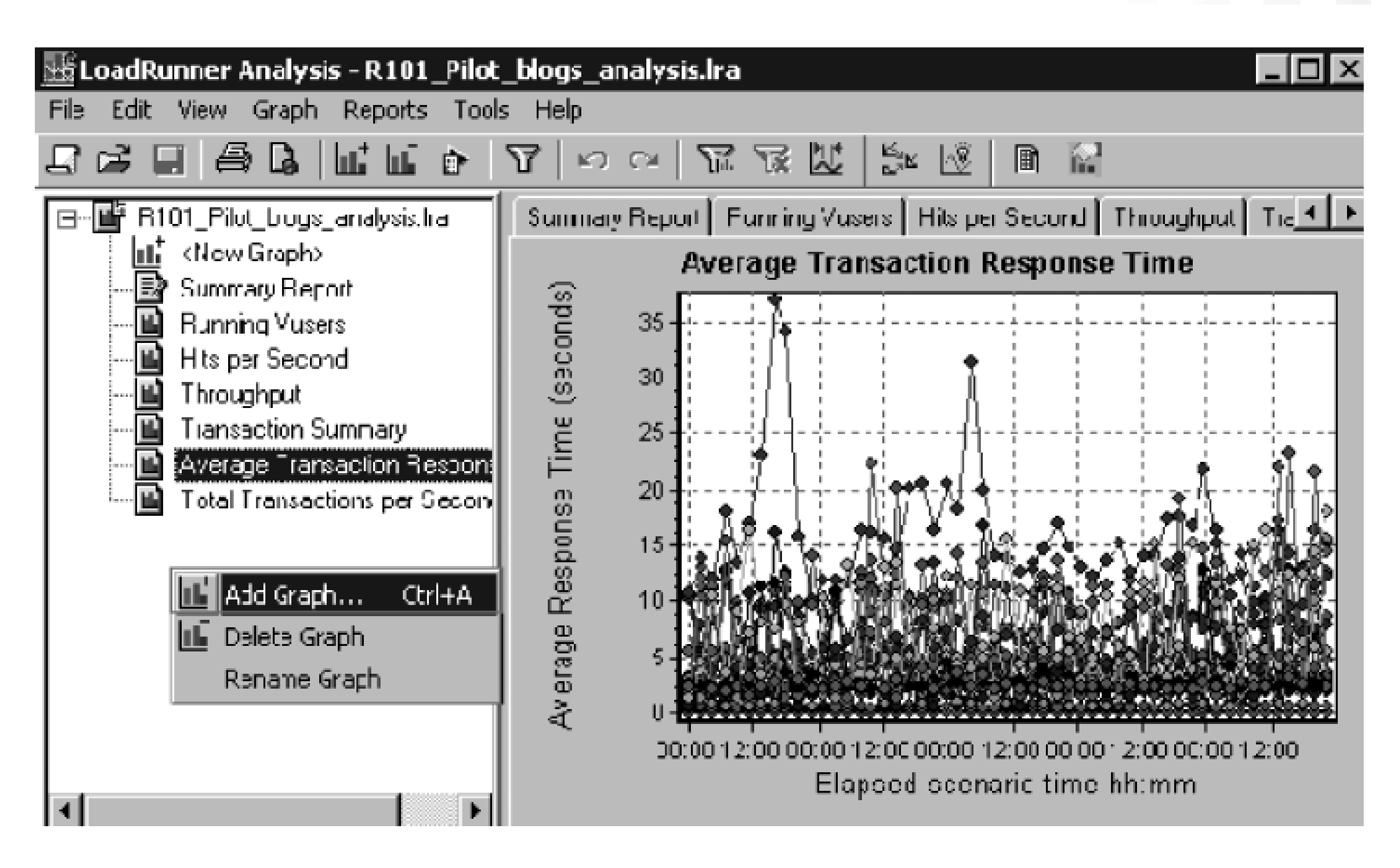


图 9-61　LR 添加其他资源监控图

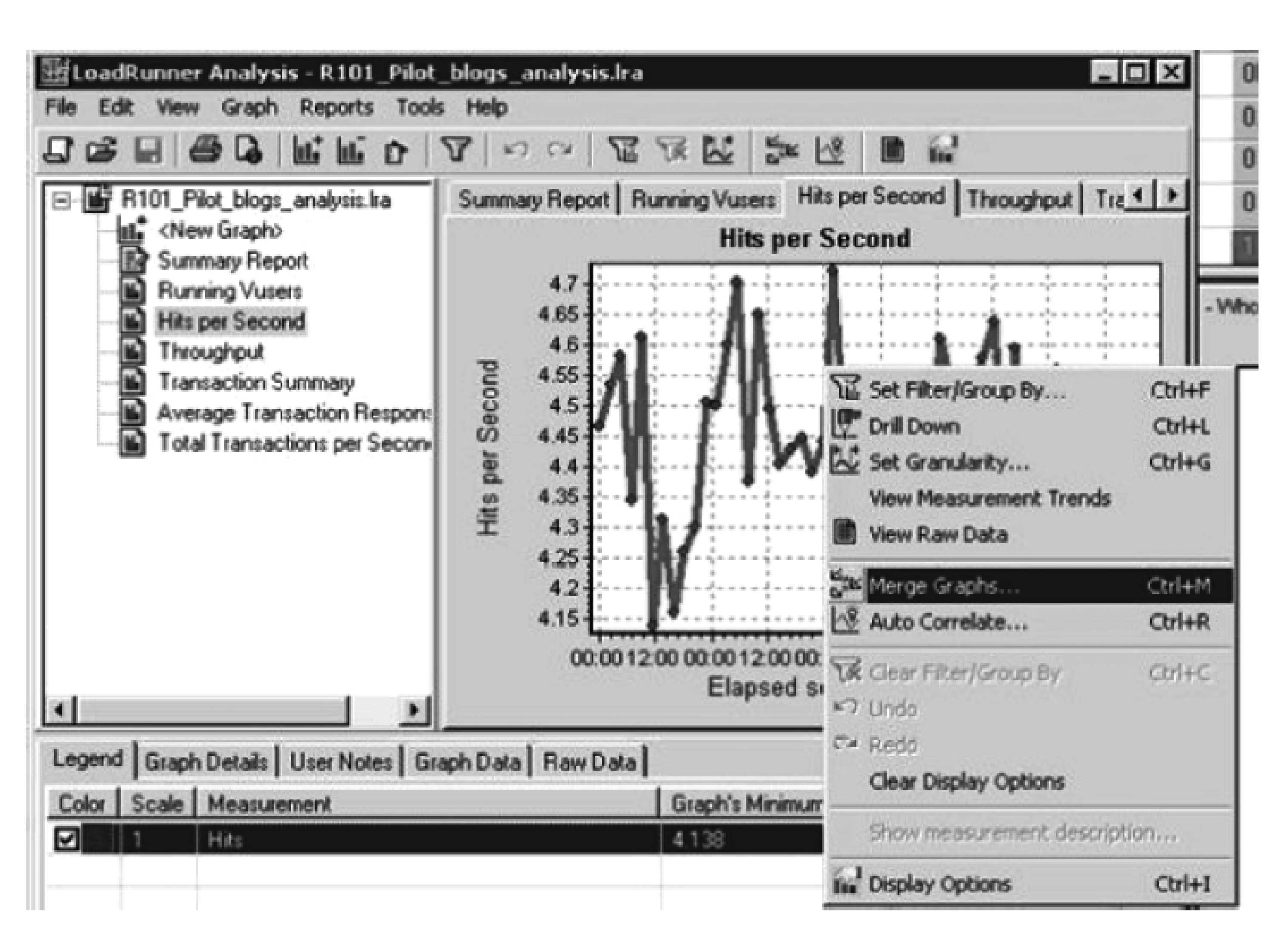


图 9-62　LR 同一次测试中多种监控图的叠加

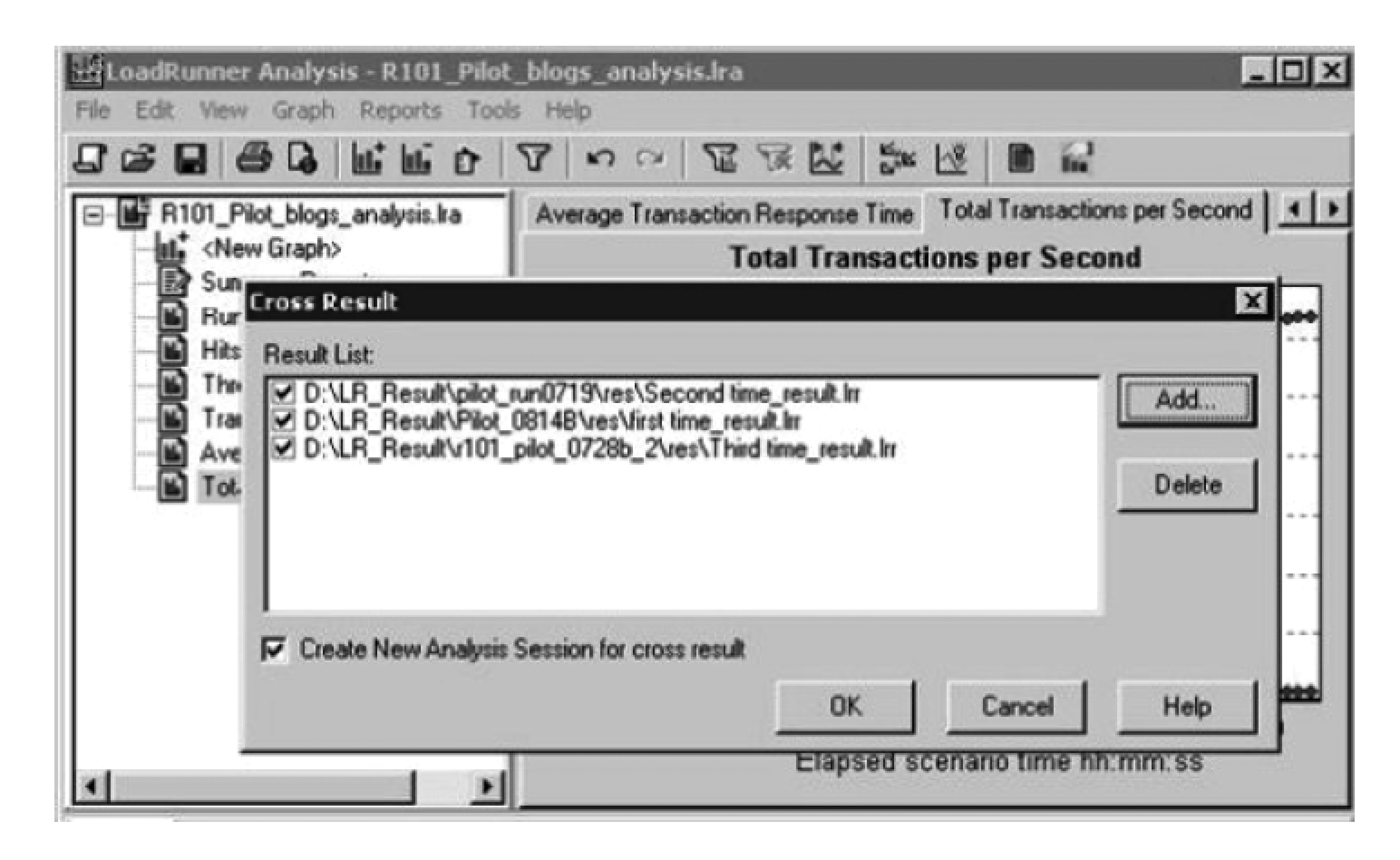


图 9-63　LR 多次历史测试结果之间的比较

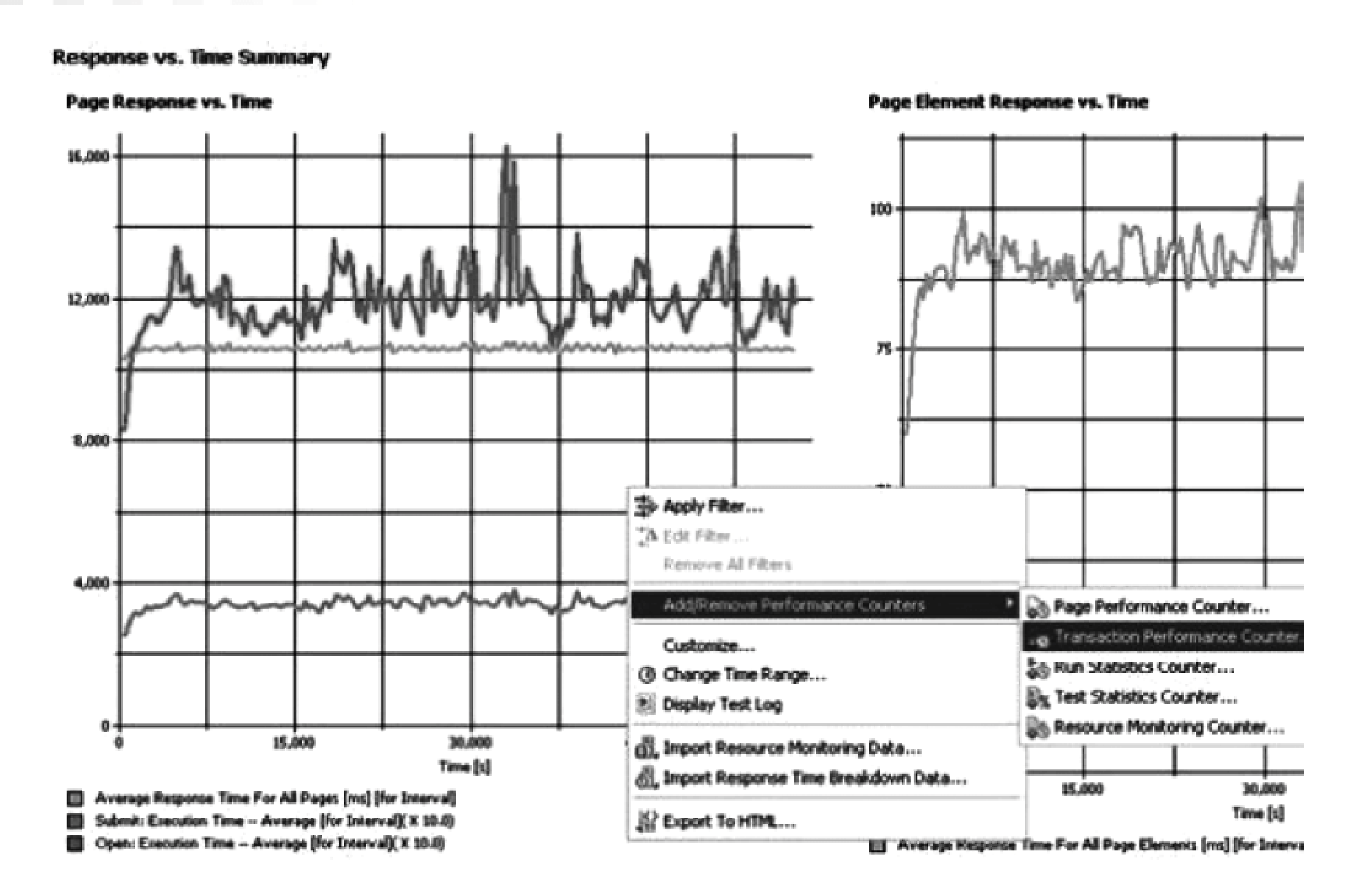


图 9-64 RPT 叠加其他资源监控图

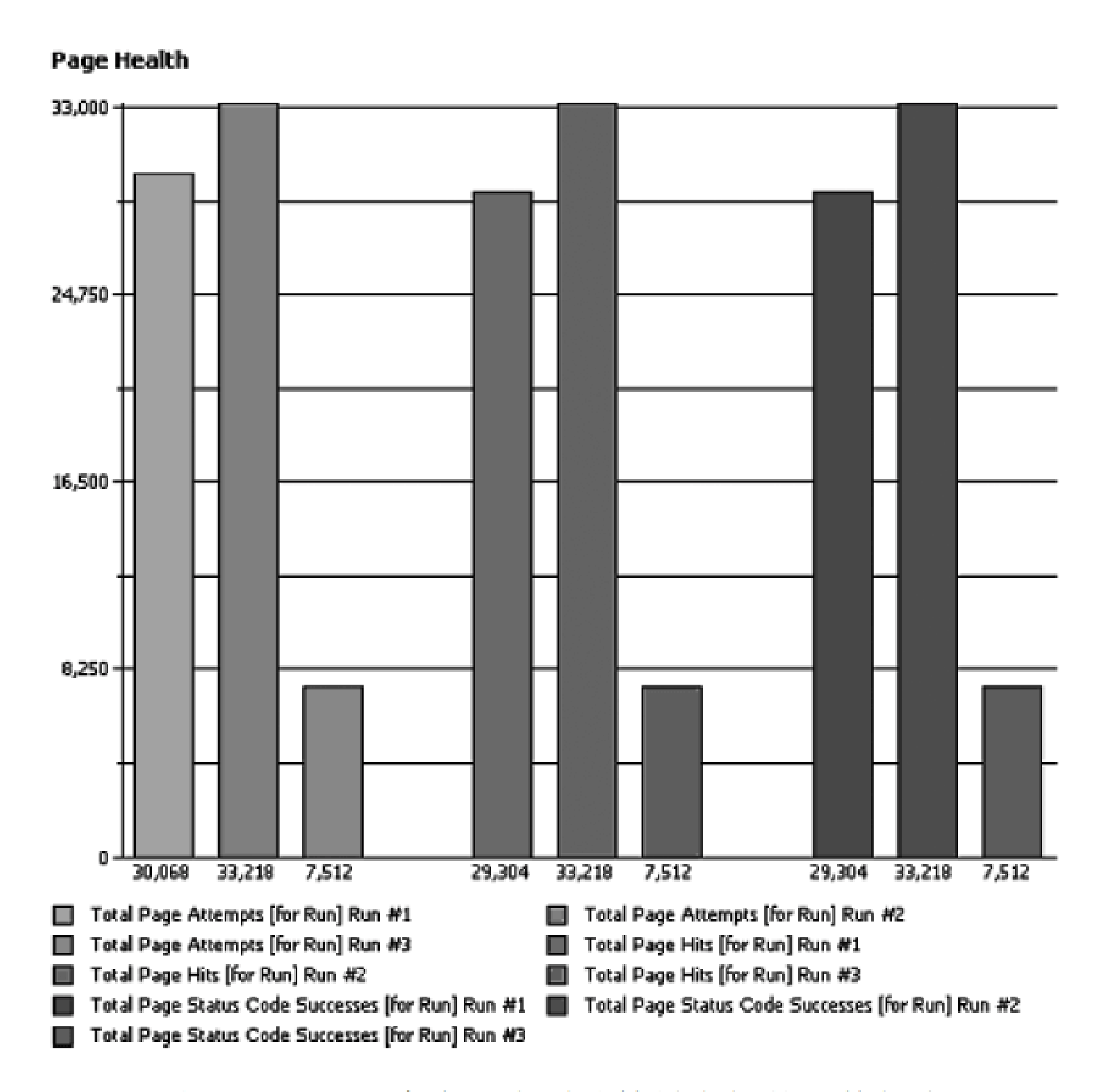


图 9-65 RPT 多次历史测试结果之间的比较报告

通过上面的对比分析，决定选择 RPT 作为 MMIS 系统的压力测试工具。下面，再介绍一些 Rational Performance Tester 的实用技巧。

1. 调整日志采样频率和粒度以适应不同的测试场景

RPT 的 Schedule 提供了测试数据的采样频率和采样粒度的配置，以适应不同的测试场景。在做长时间测试时，在 Schedule 的配置面板中的 Statistics 选项卡(图 9-66)中通过适

当增加日志采样间隔时间、日志级别和采样用户量,降低 RPT 的压力,避免因采样数据量过大,使内存耗尽,导致 RPT 无法响应。

图 9-66　Statistics 配置

在测试过程中,RPT 会记录测试中的请求数据和响应数据,以便在测试之后查看测试过程中的数据和错误信息。对于长时间测试,会产生大量的请求和响应数据,这些大量的日志信息会让 RPT 不堪重负。在 Schedule 配置部分的 Test Log 选项卡中提供了日志记录的级别设置,对于长时间的测试,推荐使用如图 9-67 所示的配置,记录错误和警告信息的级别为 All,而对于其他信息则只记录 Primary Test Actions 即可。

图 9-67　LOG 配置

2. 通过 Custom Code 实现多条测试数据的随机读取

在 RPT 中通常采用 DataPool 的形式作为少量测试数据(例如,模拟多个用户登录所用的多个用户名密码信息)的输入。不过 DataPool 的读取方式为顺序读取。如果对于输入数据需要随机读取,则可以通过 Custom Code 来实现。其实现方式可以是将测试数据存为"*.cvs"等格式文件,然后通过文件操作,并根据随机数从文件中读取内容作为测试输入数据。

3. 使用超大测试数据集文件

对于在测试过程中存在少量测试数据分次作为测试输入数据时(例如,模拟多个用户登录所用的多个用户名密码信息),通常采用 DataPool 的形式,但是在测试数据量较大时该方法就不再适用,因为对于 DataPool 中的数据,测试开始时就被全部加载到内存,如果测试数

据量过大，这种方式会造成大量的内存浪费。

这种情况可以通过 RPT 提供的扩展功能 Custom Code 来定制代码，达到在测试过程中在需要的时候再去加载所需的内容，其实现也可以采用文件操作的方式。

9.3.3 获取 RPT License Key

正式使用 RPT 前，需要获得官方授权的 License Key。涉及的要点内容介绍如下。

1. 进入 LKC

(1) 打开 License Key Center(LKC)的网址：http://www-306.ibm.com/software/rational/support/licensing/。

(2) 单击 IBM Rational License Key Center。

(3) 在新页面中单击 Continue 按钮。

2. 加入账户

(1) 在 Rational License Key Center 页面中，单击链接 Don't have a password。

(2) 填写表格。Sales Order Number 格式：00 *** ，Site Number 格式："000 *** 。Sales Order Number 和 Site Number 可以在授权书(Proof of Entitlement)上找到。单击 Submit 按钮。

(3) 用户会收到一封写有 LKC 临时密码的邮件。在第一次进入 LKC 时，必须修改密码。

3. 产生 license

成功进入 LKC 后，可以按以下步骤产生 license。

(1) 在左边的选项中，单击 Get Keys。在 Get Keys 页面中选择需要的 license 种类。

(2) 在 Select license key 页面上，在所选择的 license 前打勾，并且单击 Generate 按钮。

(3) 在 Quantities and host information 页面，请填写必填部分。

① 在 Number of keys to generate 中写下数字。最多可用的数目为 Number of keys available 下的数字。

② 在 server host ID 旁的下拉菜单里为 license 所在的机器选择平台。

③ 在已经安装产品的 license Server 机器上运行：开始→Run→写下 hostinfo。把信息复制到 Host ID 和 Host Name 里。

④ 单击在页面下方的 Generate 按钮。

9.3.4 RPT 更新

打开 IBM Installation Manager，如图 9-68 所示。

启动 IBM Installation Manager，界面如图 9-69 所示。



图 9-68　打开 IBM Installation Manager

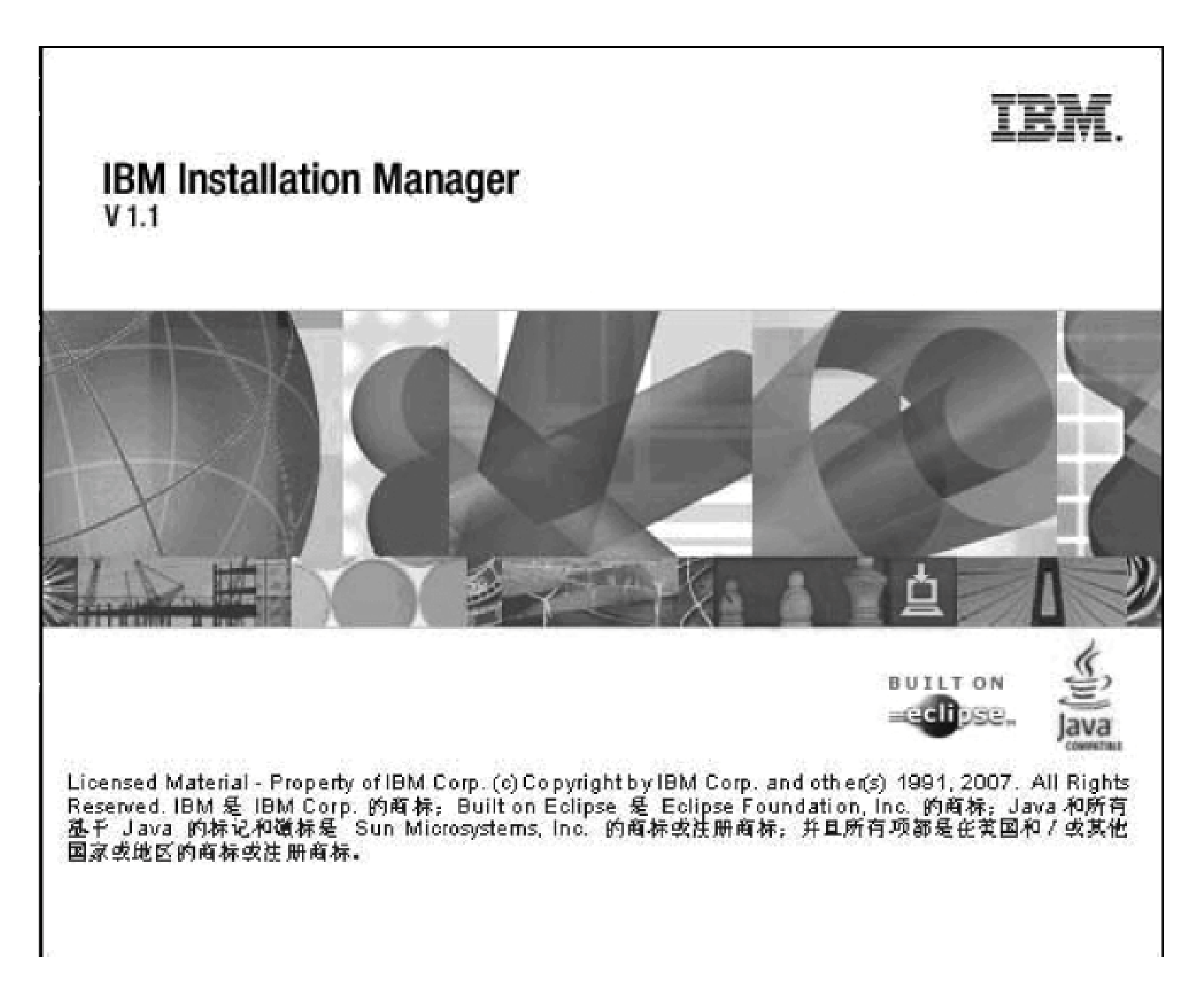


图 9-69　IBM Installation Manager 的启动界面

IBM Installation Manager 的主界面如图 9-70 所示。

单击 IBM Installation Manager 的“文件”→“首选项”菜单，如图 9-71 所示。

取消勾选“在安装和更新过程中搜索服务存储库”复选框，在弹出的“首选项”窗口中单击“添加存储库”按钮，如图 9-72 所示。

在弹出的“添加存储库”对话框中单击“浏览”按钮，如图 9-73 所示。

弹出“选择存储库”对话框，如图 9-74 所示。

选择“存储库”的位置，如图 9-75 所示。

图 9-76 是 IBM Installation Manager 升级时使用的存储库文件，选择后单击“打开”按钮。



图 9-70 IBM Installation Manager 的主界面



图 9-71 单击“首选项”

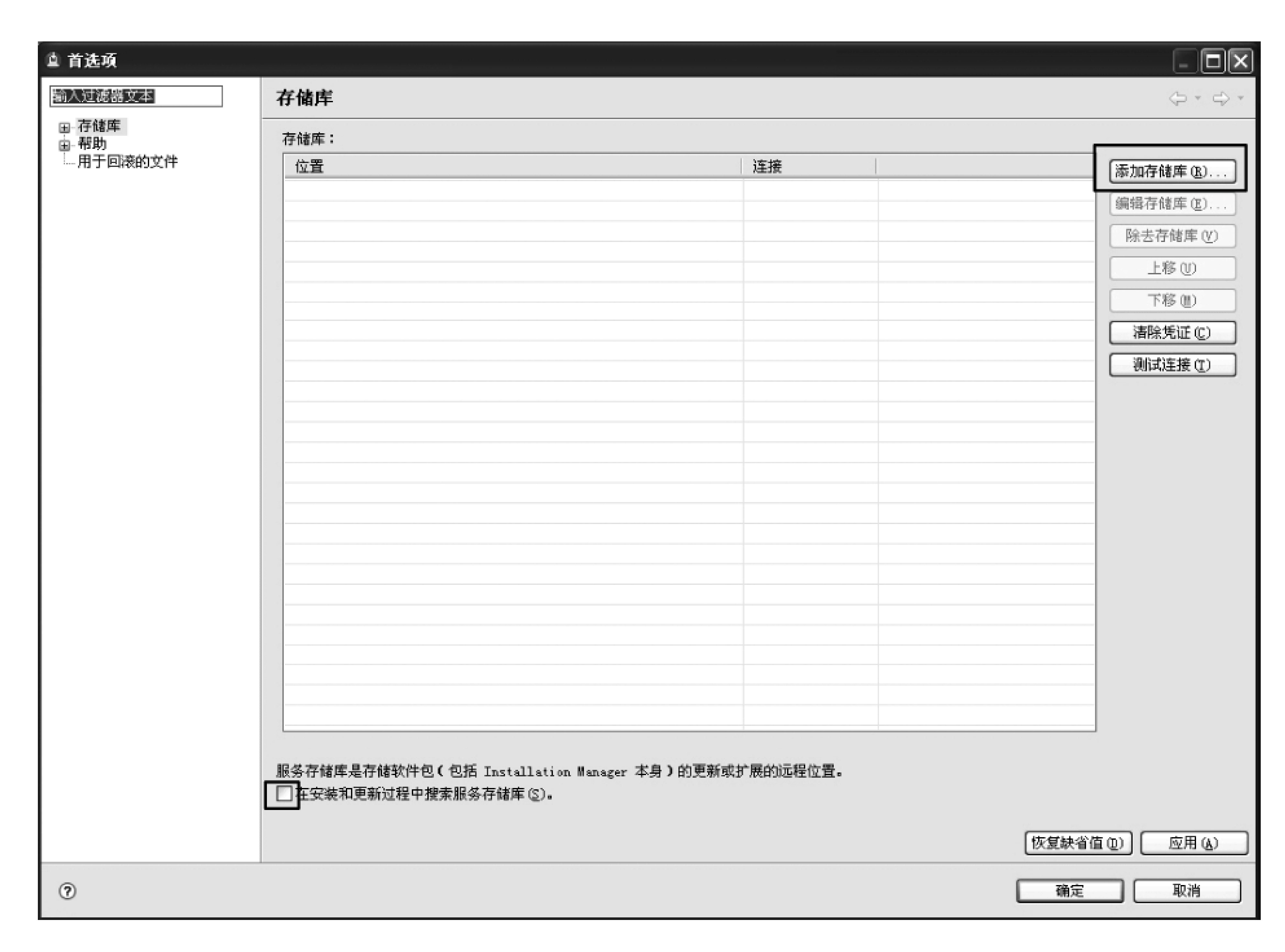


图 9-72　去掉搜索服务存储库并添加存储库

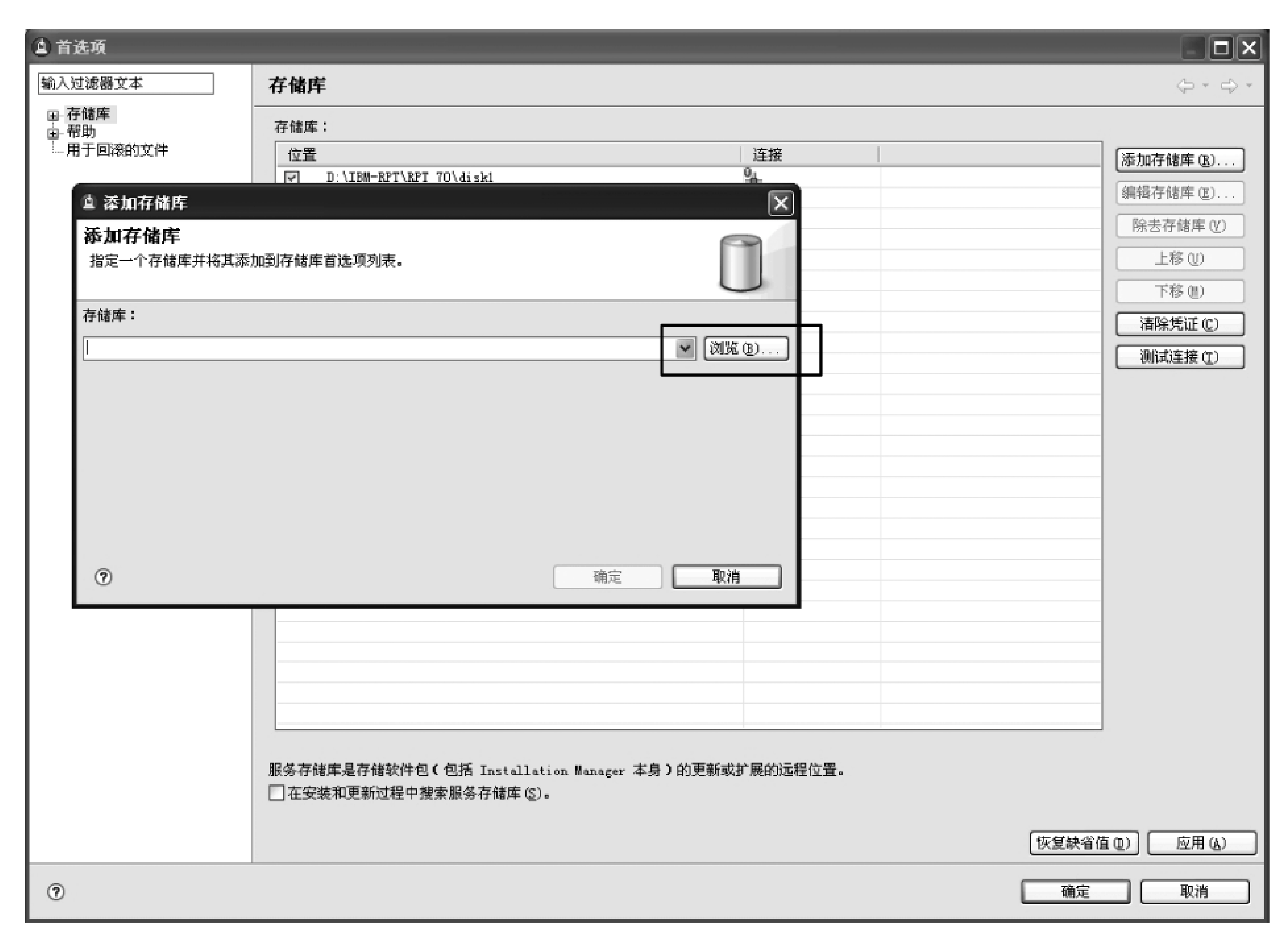


图 9-73　单击"浏览"按钮

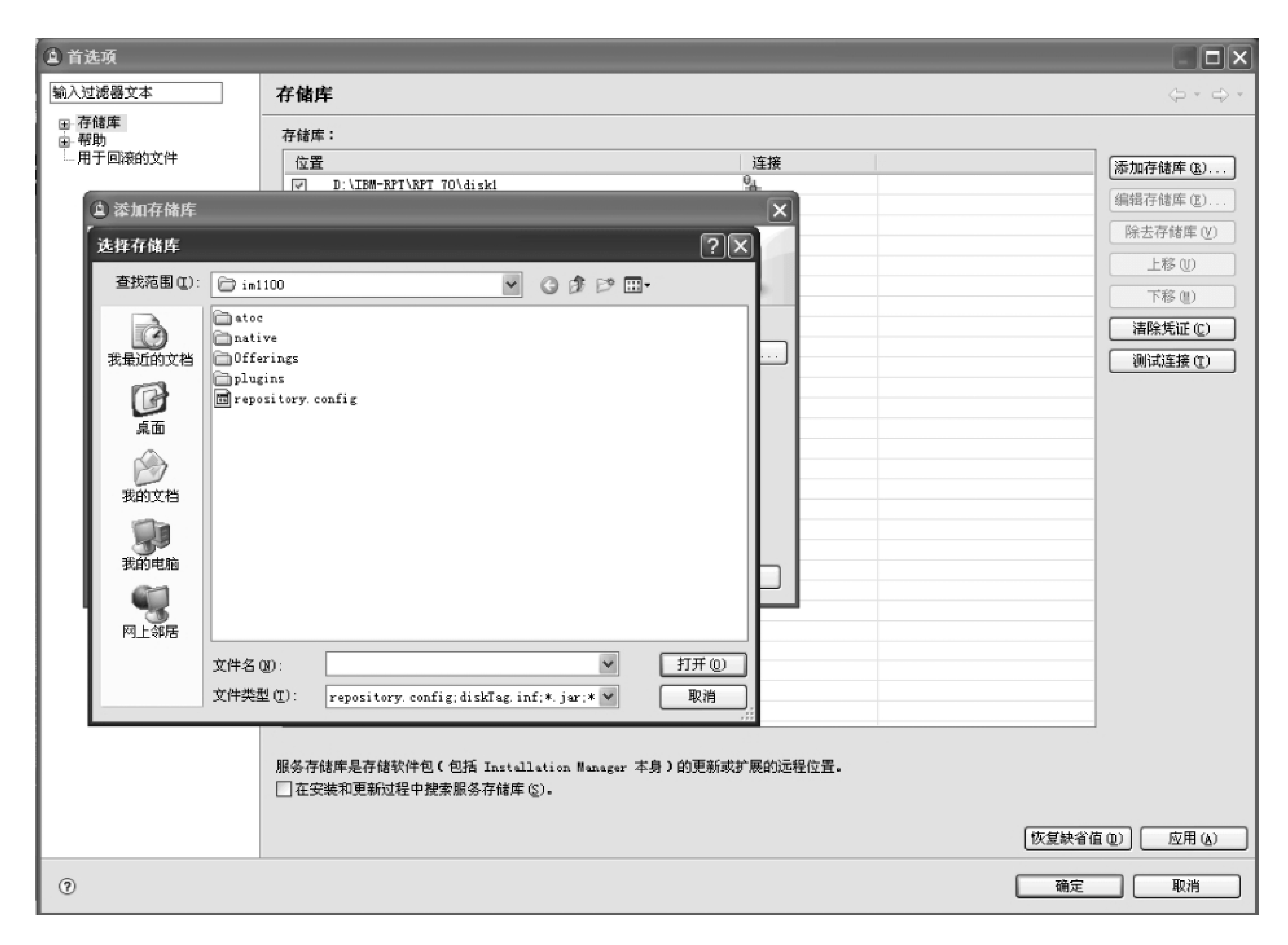


图 9-74 选择存储库

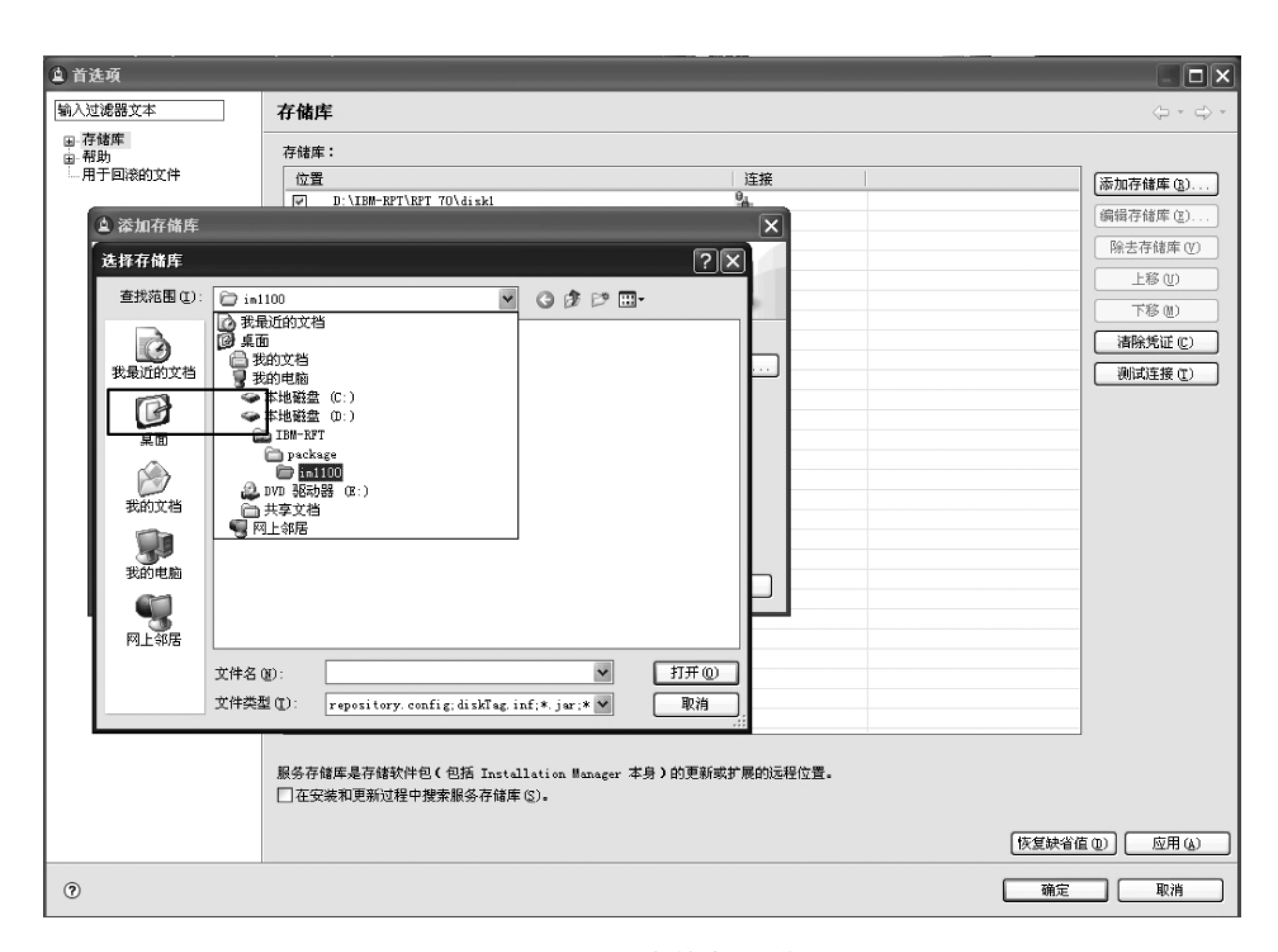


图 9-75 选择“存储库”的位置

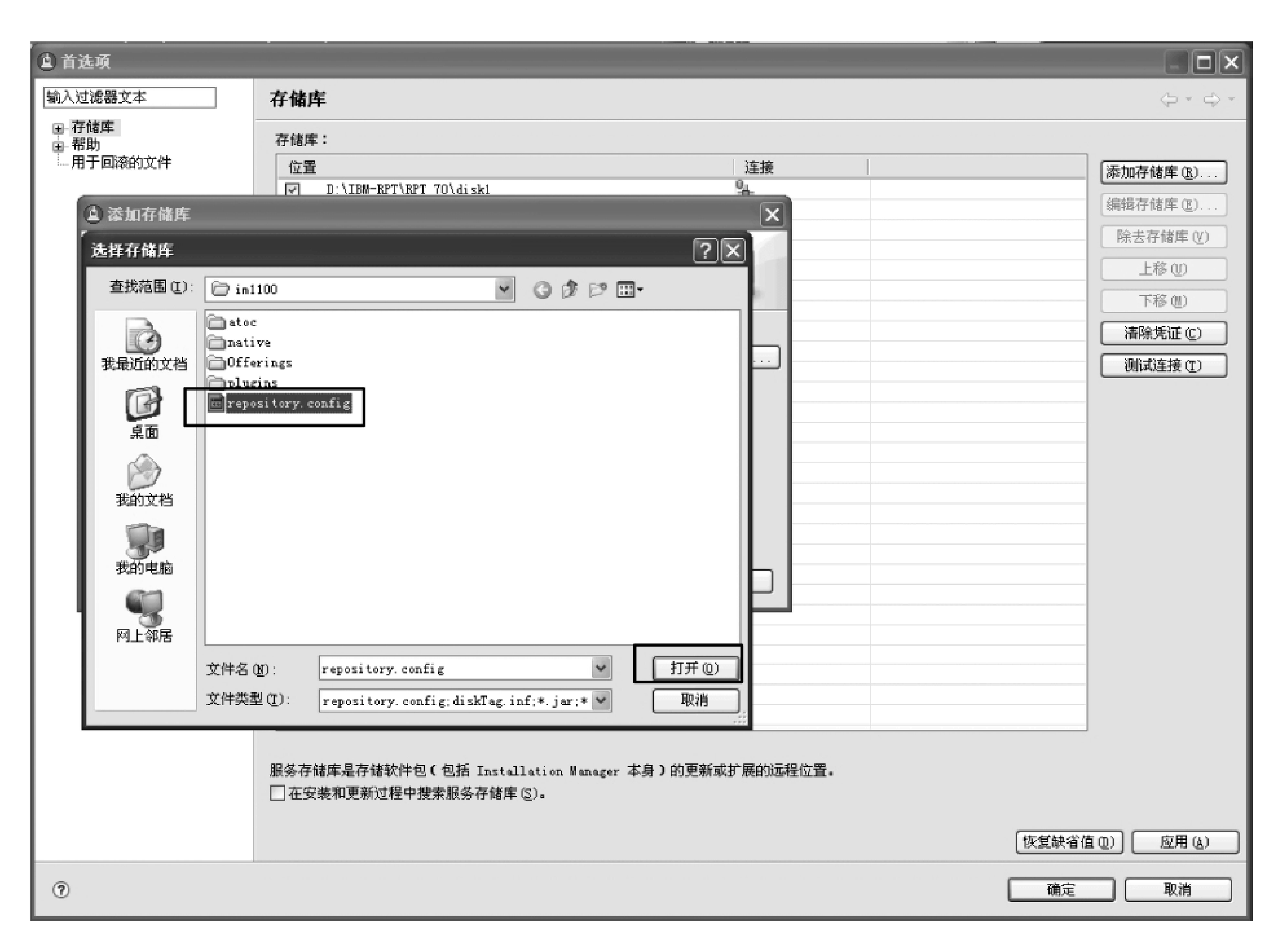


图 9-76　升级时使用的存储库文件

“存储库”的路径添加完成后单击“确定”按钮，如图 9-77 所示。

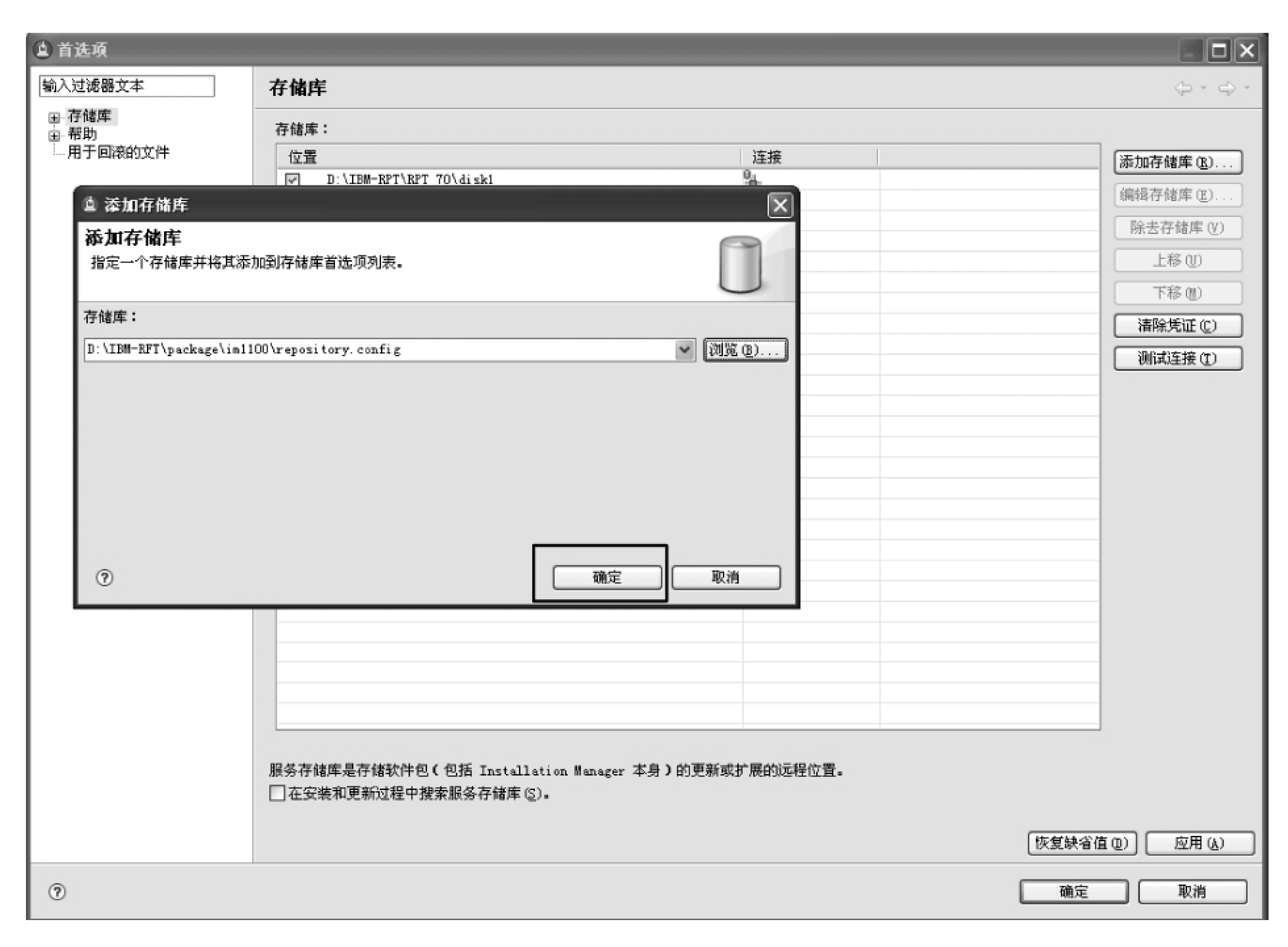


图 9-77　确定“存储库”的添加路径

相应地添加需要的“存储库”，例如 RPT 7.0 的安装存储库和 RPT 7.1 的升级存储库，如图 9-78 所示。

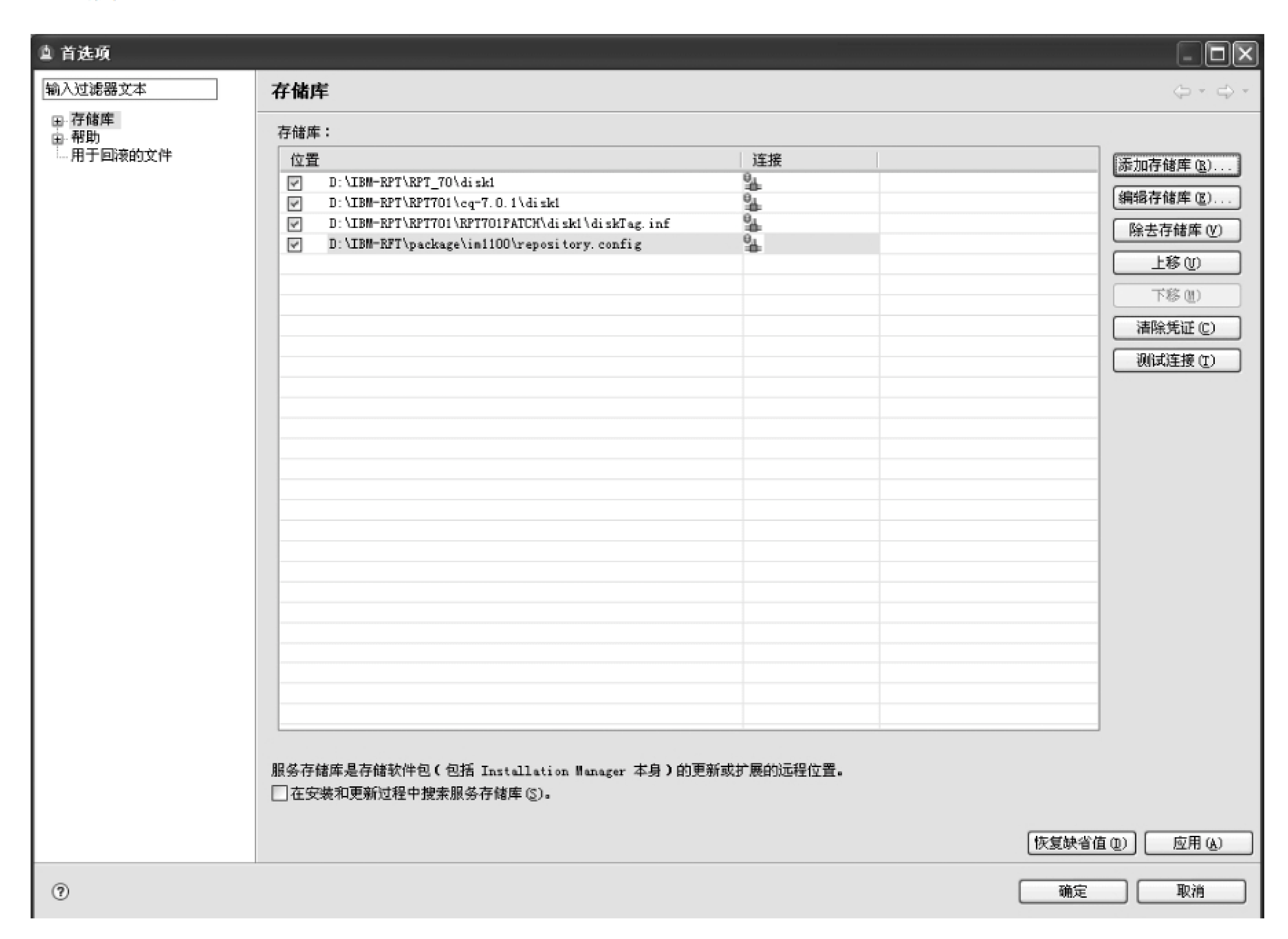


图 9-78　添加需要的“存储库”

单击“更新软件包”，如图 9-79 所示。



图 9-79　更新软件包

选择需要更新的软件包，单击“下一步”按钮，接下来的步骤基本上都是单击“下一步”按钮的操作，直到安装或更新完成，如图 9-80 所示。

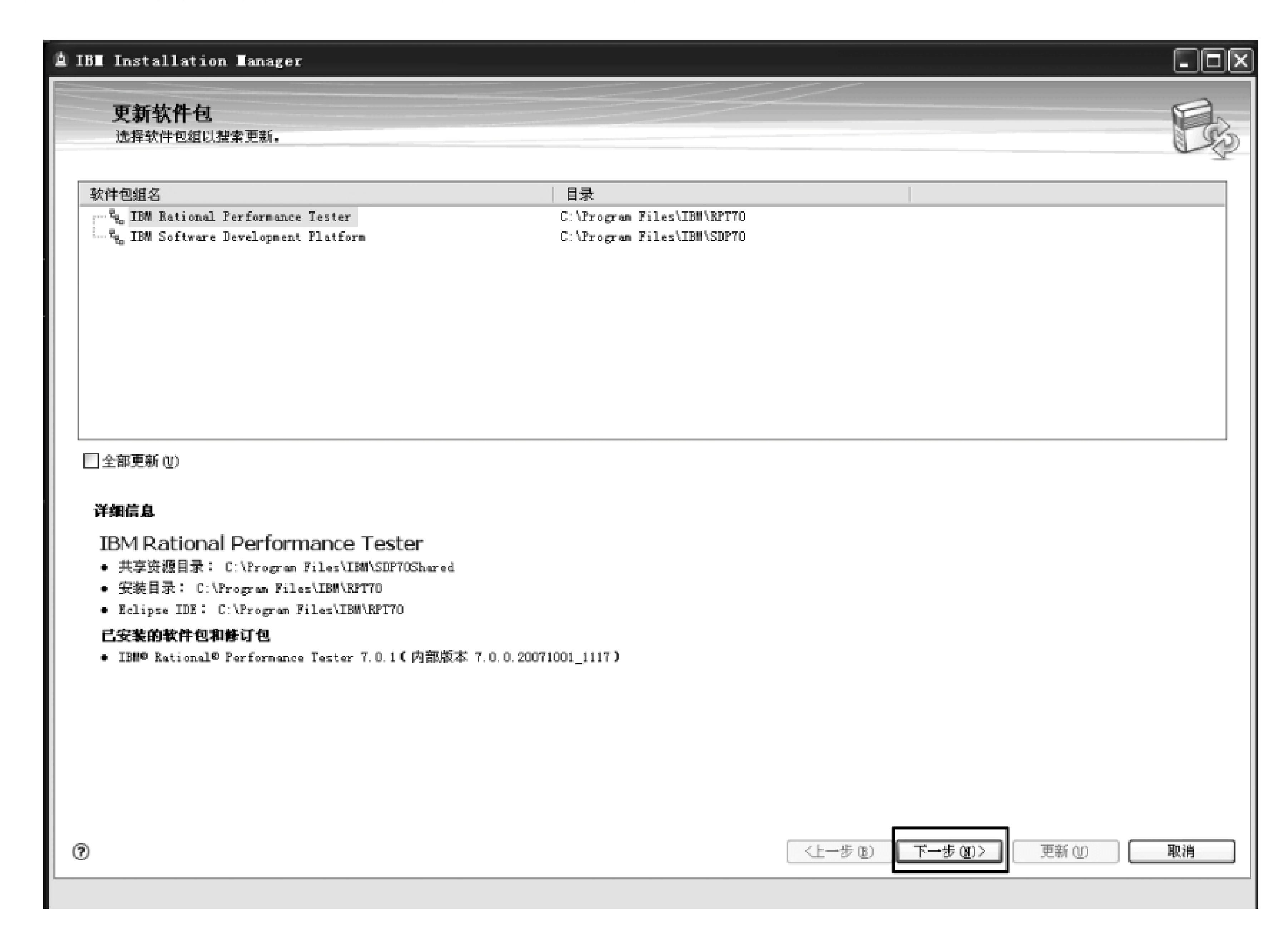


图 9-80　单击“下一步”按钮

在下一个页面下载 license(.upd)到 license server 上。在 license server 上打开此.upd 文件，如图 9-81 所示。

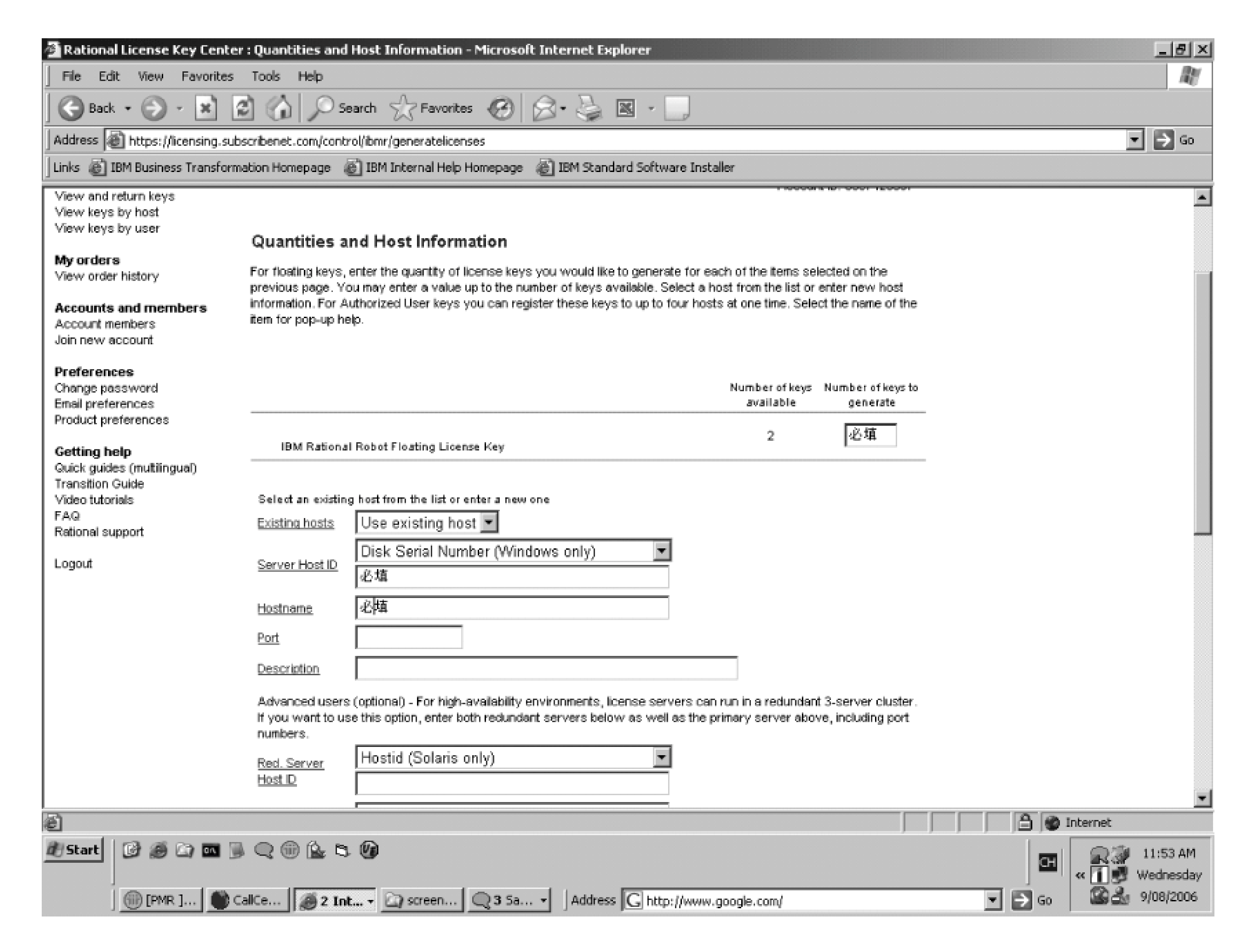


图 9-81　下载 license

9.3.5 RPT 创建项目

(1) 新建 D:\StressTestProject 文件夹，如图 9-82 所示。

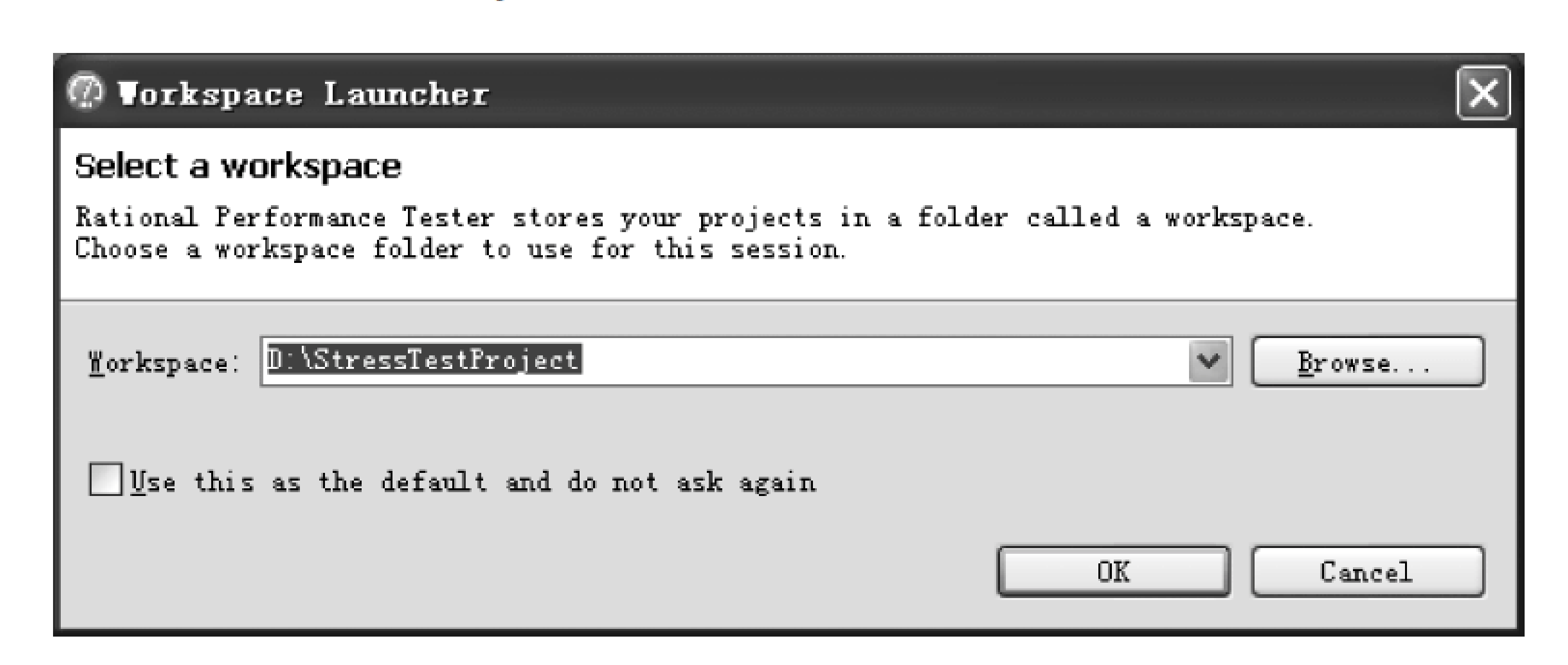


图 9-82　新建文件夹

(2) 打开 RPT，如图 9-83 所示。

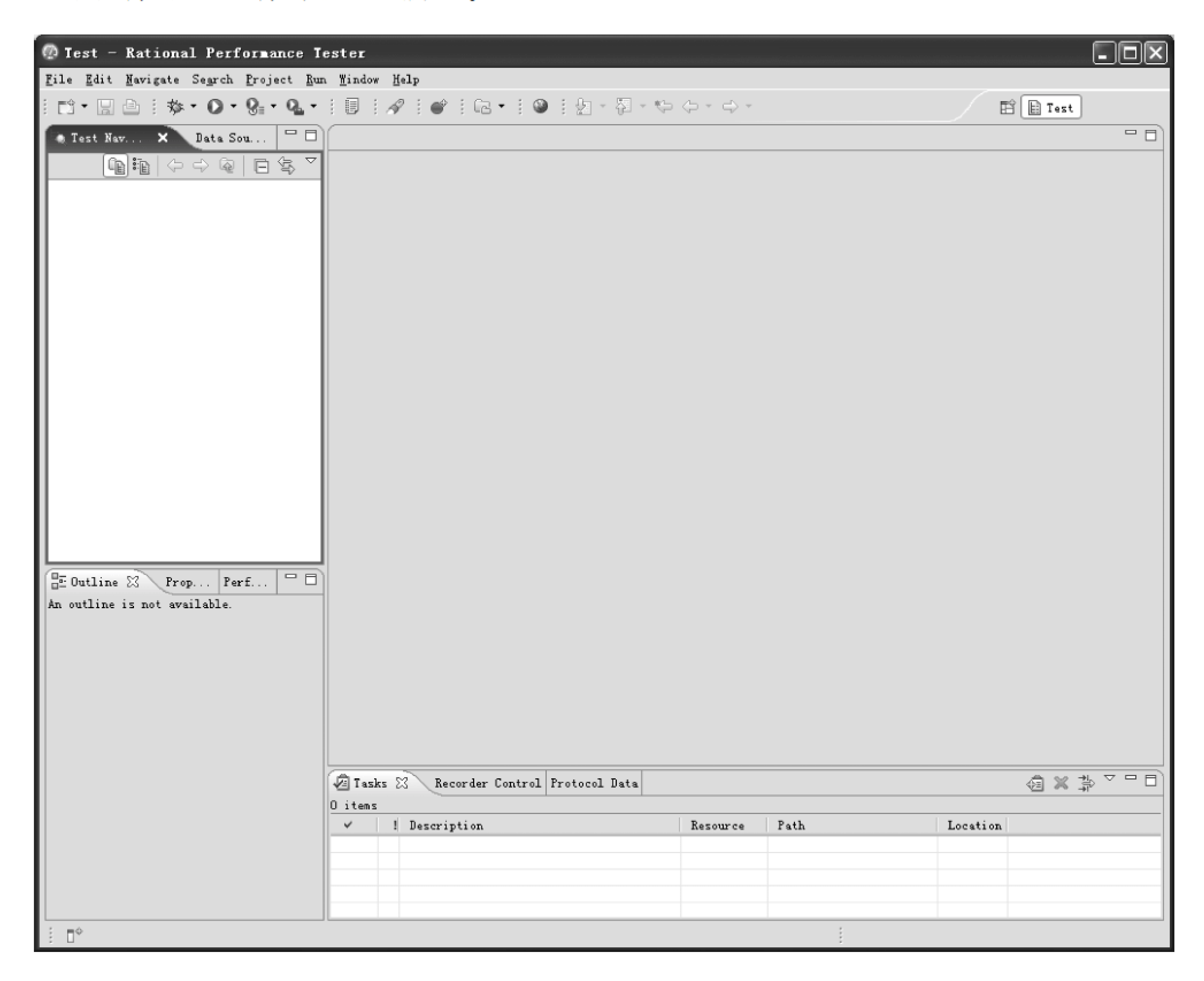


图 9-83　RPT 界面

(3) 选择菜单 File→New→Performance Test Project 来创建一个新的 RPT 项目，如图 9-84 所示。

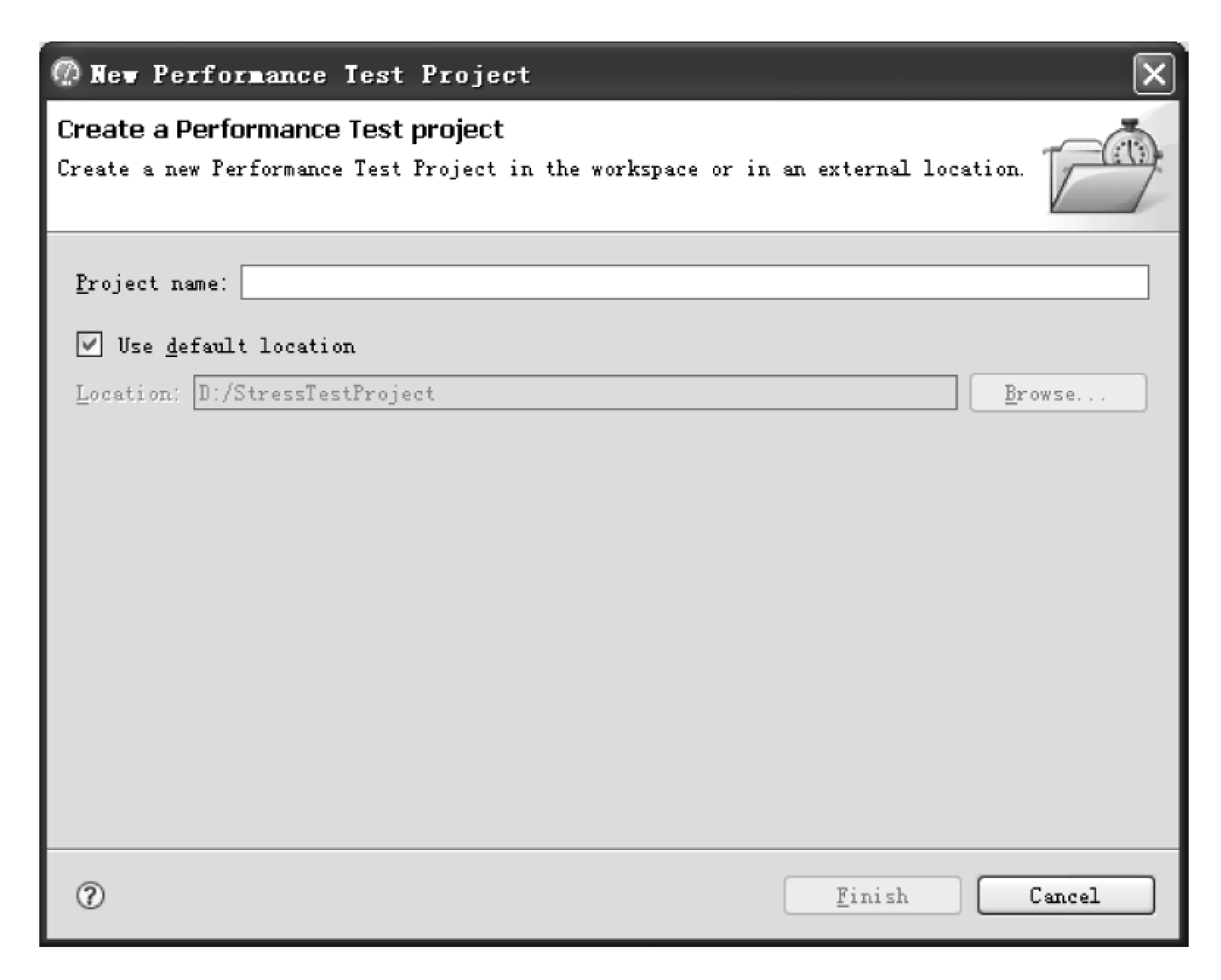


图 9-84　创建 RPT 项目

（4）输入“mmisTestProject”，然后单击 Finish 按钮，如图 9-85 所示。

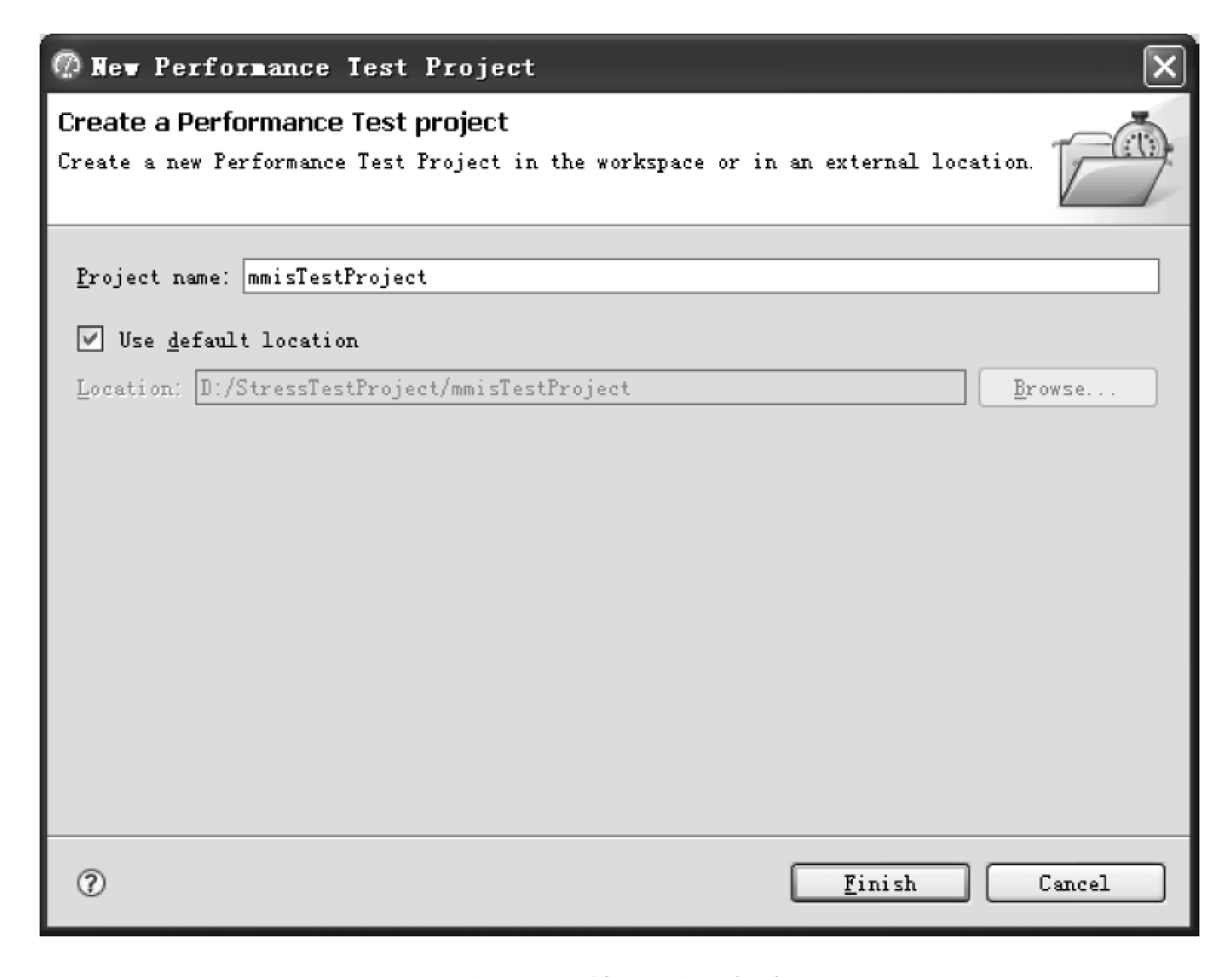


图 9-85　输入项目名称

（5）打开 D:\StressTestProject 文件夹，然后把 1BM 发出的 RAR 包的内容解压，复制以下3 个高亮的目录到 D:\StressTestProject 文件夹，如图 9-86 所示。

（6）返回 RPT 界面，右击，选择 Refresh 选项，如图 9-87 所示。

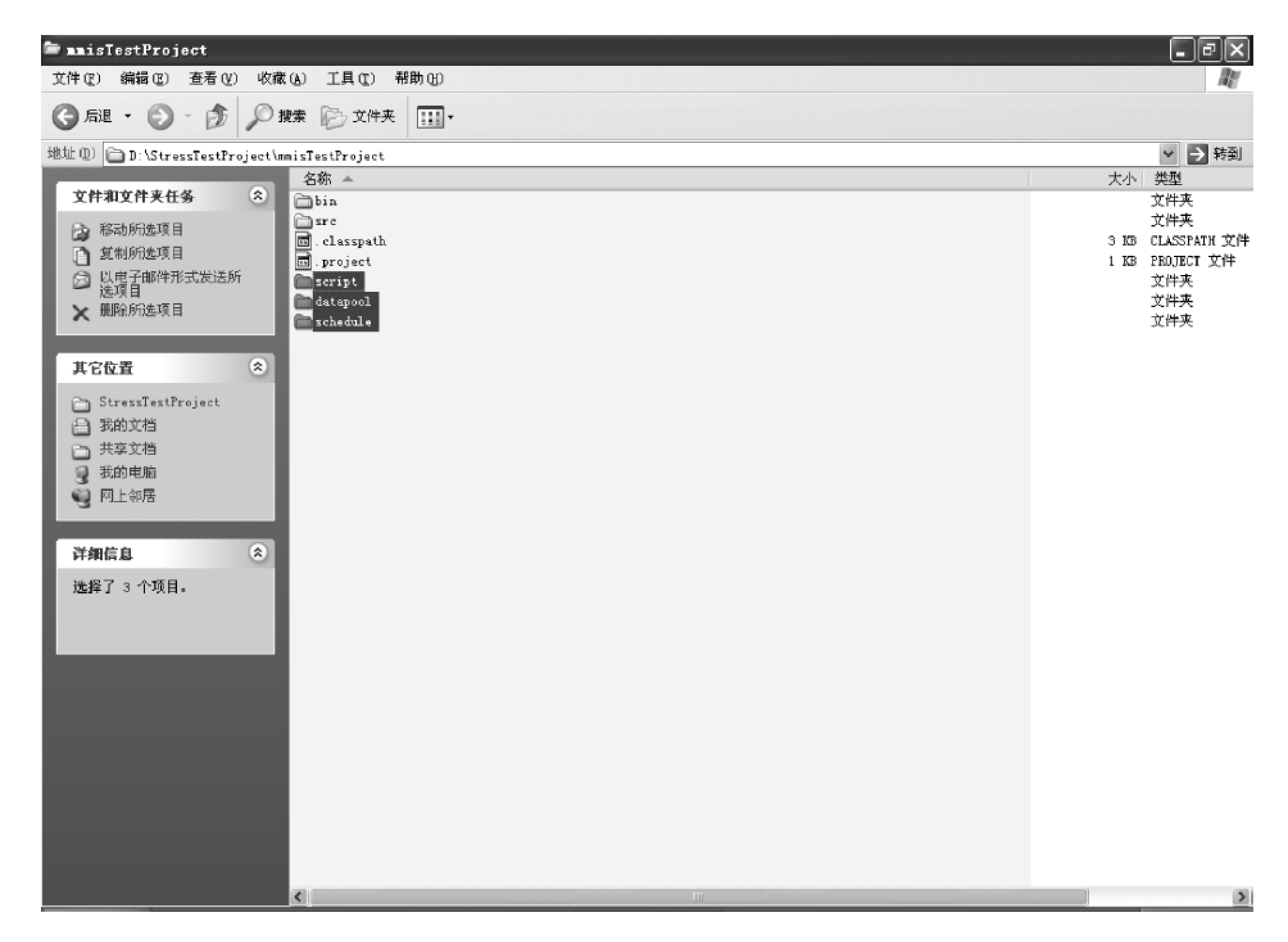


图 9-86　项目文件夹

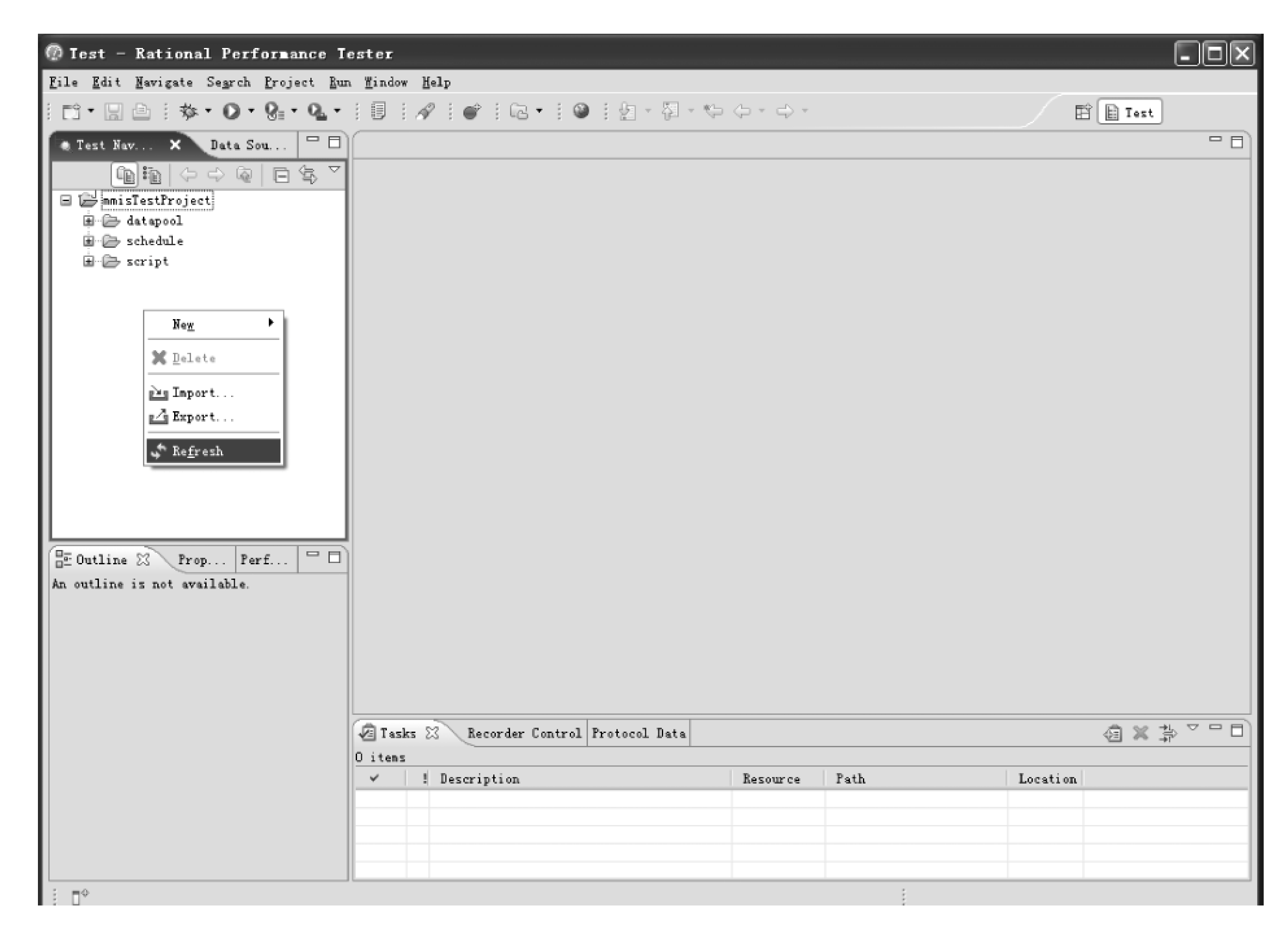


图 9-87　刷新

9.3.6 系统压力测试

下面从压力测试的范围、测量标准、环境及所需的数据等几个方面介绍使用压力测试工具 IBM Rational Performance Tester 7.0 完成对维护管理信息系统(MMIS)压力测试的情况。

1. 压力测试的目标

压力测试的目标是要确保 MMIS 系统能在不同的载荷情形下符合津澳地铁与电讯盈科所商定的性能要求。在受控测试环境中实现 MMIS 中的关键业务功能将会在生产环境中所预计的正常及最高载荷情况下有效及高效地执行,从而为项目的顺利完成确立一定的信心。

主要的目标是确保:

(1) 性能测试在即将的生产环境中进行。

(2) 性能测试涵盖最为关注及已议定的事务处理,并以津澳地铁同意的功能性测试场景为基础。

(3) 性能测试要确保 MMIS 能满足以下需求。

① 在线创建/更新/删除/获取一条记录:<5s。

② 在线创建/更新/删除多条记录:<15min。

③ 根据用户指定标准进行的数据获取:<1min。

④ 超过这些基准的事务处理应安排以批量模式运行,或微调系统设计/程序逻辑以达到指定的响应时间。

2. 压力测试的范围

已识别目标功能是整体应用用途中的关键领域。整体测试场景基本上是来自于功能及系统/整合测试场景。

表 9-4 是已识别的关键事务处理,其中包括进行性能测试的创建/更新/删除/获取操作。

表 9-4 已识别的关键事务处理

模块	子 模 块
EI	Location
	Equipment Registration
	Fitment/Refitment
MP	Standard Job
	Standard Job Pattern
	PM Master
	Planning Workbench
	Measurement
WM	Work Order
	Work Request
	Timesheet
	Incident

以下所有列出模块的相关活动不包括在本次性能测试的范围之内，属于范围外的活动，如表 9-5 所示。

表 9-5 范围外活动

模块	子 模 块
EI	Equipment Class
	Equipment Class Group
	Equipment Structure Definition
	Equipment Structure
	Warranty & Contract
	Reporting Hierarchy
	Parts Catalog
SA	Maintain Personalization
	Change Password
	Organization
	Work Group
	Staff/Contractor
	User Role
	Access Group
	Security Policy
	User Account
	System Parameter
	System Period
	Batch Job
	Upload Interface
	Generic Code
	Work Nature
	Labor Rate
	Maintain Stock Code and Standard Unit Cost
	Document Upload
	Document Housekeep
	User Defined Field
	Home Page
PA	Performance Standard
	Budgets
	Cost
	All Reports

3. 测试场景

如表 9-6 所示，主要的测试环境为 50 个在线用户将会同时在不同的模块执行不同的动作。

(1) 12 个在线用户将会在 EI 模块执行操作。

(2) 20 个在线用户将会在 MP 模块执行操作。

(3) 18 个在线用户将会在 WM 模块执行操作。

表 9-6　测试场景表

类别	模　　块	功　　能	用户数目
EI	Location		6
		Save	
		Search	
	Equipment Registration		3
		Save	
		Search	
	Fitment/Refitment		3
		Confirm Fitment/Refitment	
		Movement	
MP	Standard Job		6
		Save	
		Search	
	Standard Job Pattern		4
		Save	
		Search	
	PM Master		6
		Create PM Master	
		Search	
		Tabulate	
	Planning Workbench		2
		Job search	
		Work order generation	
	Measurement		2
		Search reading	
		Measurement statistics calculation	
		Measurement statistics search	
WM	Work Order		12
		Quick Form/Save	
		Quick Form/Change Status	
		Quick Form/Find	
		Quick Form/Search	
		Full Form/Save	
		Full Form/Find	
	Work Request		
		Save	2
		Create work order	
		Search	
	Timesheet		2
		Save	
		Search	
	Incident		2
		Save	
		Search	

4．压力测试方法

整体压力测试阶段的流程将会如图 9-88 所示。

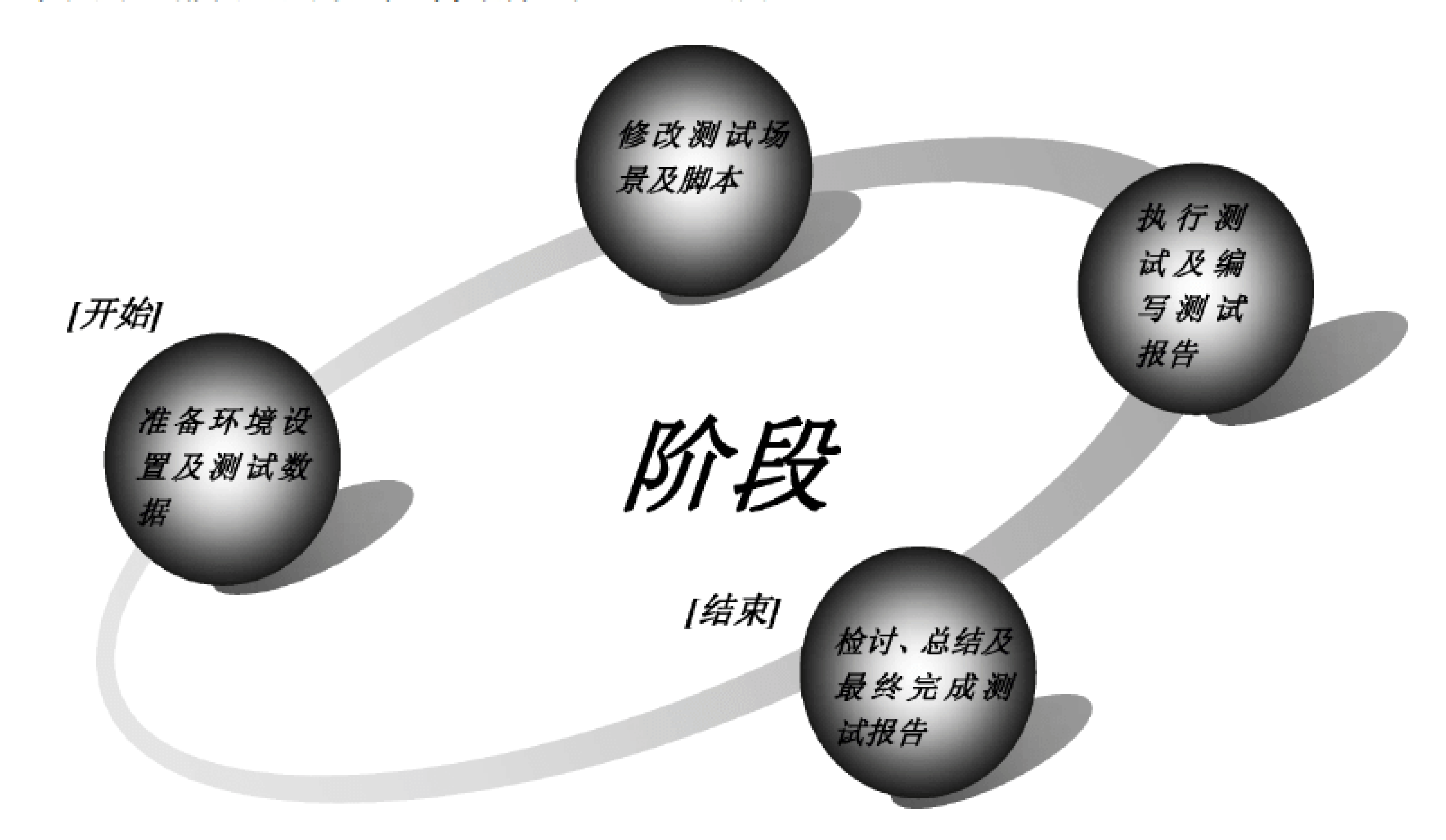


图 9-88　压力测试流程

压力测试有 50 个模拟用户模拟加载过程。所有压力测试场景会逐一执行并记录测试的结果。

以下组件是 OLTP 事务处理的性能测试。

(1) 自动测试工具 Rational Performance Tester 用于模拟多用户的事务处理。

(2) 性能统计数据会按照以下方法收集。

① 应用：由 Rational Performance Tester 提供性能报告。

② 数据库：由 Oracle 性能工具获取数据库性能测量数据。

③ 服务器：一套一般 UNIX 指令(如 vmstat 和 iostat)及 Rational Performance Tester 用于获取操作系统的水平测量数据。

1) 进入标准

(1) 压力测试环境应用所有所需的应用程序及数据库补丁。

(2) 按照应用配置文档配置 MMIS 模块。

(3) 压力测试服务器的日期及时间是实际的日历日期时间。

(4) 主数据及事务处理数据成功加载到压力测试环境。

(5) 审阅及完成测试数据、测试场景及脚本的准备工作。

(6) MMIS 压力测试环境是可操控的，并获得所需的程序/报告。

(7) 压力测试方案经由津澳地铁及电讯盈科双方商定。

2) 退出标准

(1) MMIS 压力测试环境是可操控的，并获得所需的程序/报告。

(2) 数据成功加载到压力测试环境。

(3) 已执行测试循环的所有测试用例。

(4) 有实际的书面测试结果作为测试的证明。

(5) 获取并签署压力测试实际结果。

(6) 已成功测试 MMIS 的所有测试循环、子循环及测试用例。

（7）压力测试结果必须达到第 1.1 节所述目标。
（8）津澳地铁已同意并签署以上标准的任何例外情况。

5. 测试环境

压力测试的环境如图 9-89 所示，共有一台 HP 应用服务器用于本次压力测试，并无任何集群环境。

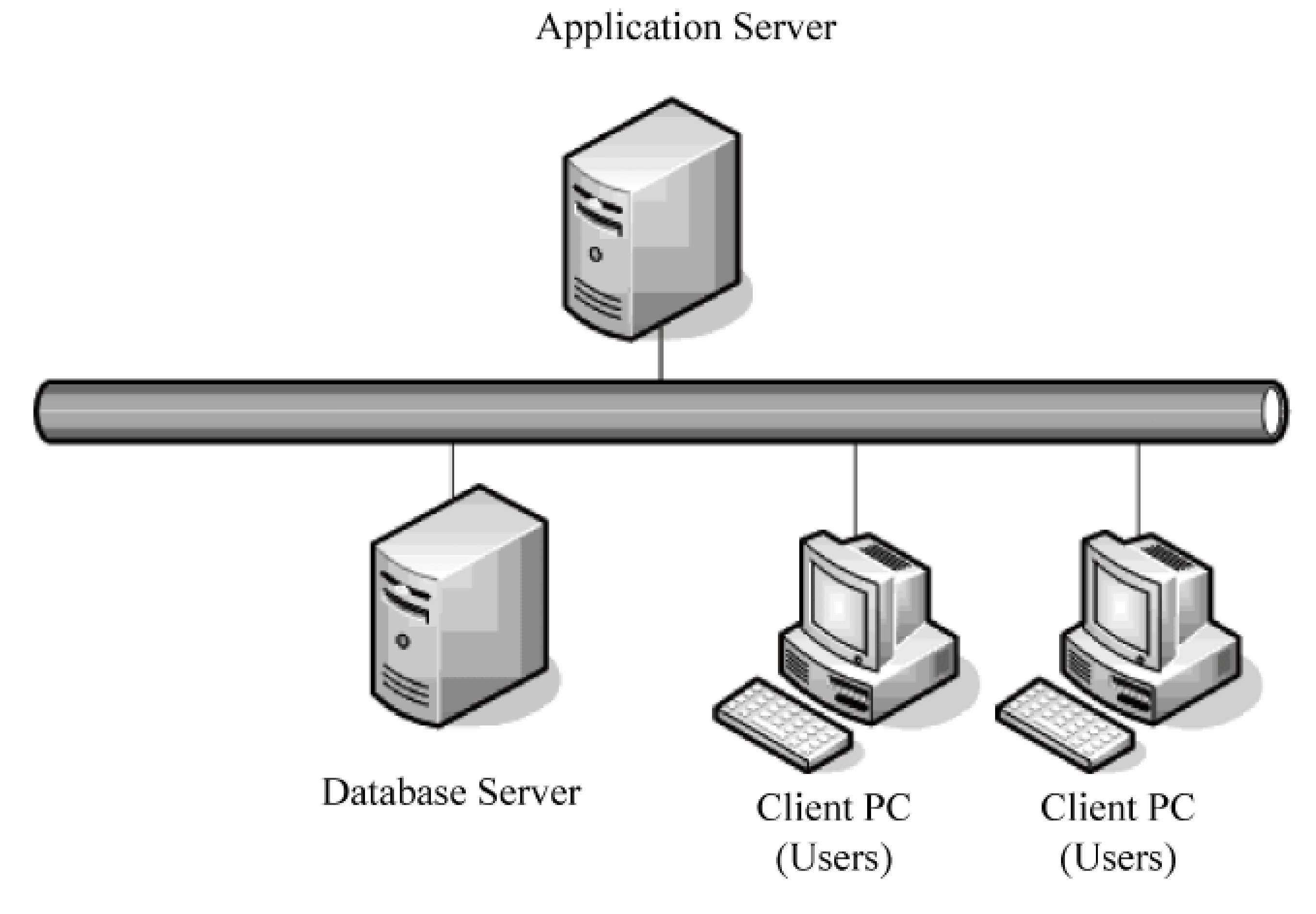


图 9-89　压力测试环境

MMIS 系统软件的版本是 V1.0 BUILD1。
应用服务器为一台 HP 服务器，其硬件配置如表 9-7 所示。

表 9-7　应用服务器硬件配置

类　　型	Application Server	类　　型	Application Server
CPU	4×4 2000 MHz	硬盘空间	136GB
内存	4GB	IP Address	10.199.0.198

应用服务器的系统软件如表 9-8 所示。

表 9-8　应用服务器的系统软件

操 作 系 统	Red Hat Linux AS 4.1
系统软件	JBoss Application Server 4.2.1.GA　x　2
Java Runtime Environment	JRE 5.0

数据库服务器为一台 HP 服务器，其硬件配置如表 9-9 所示。

表 9-9　数据库服务器硬件配置

类型	Database Server
CPU	双 CPU、Itanium® processor，2×1.6 MHz
内存	8GB
硬盘空间	100GB
IP Address	10.199.0.15

数据库服务器的系统软件如表 9-10 所示。

表 9-10 数据库服务器的系统软件

操作系统	HP UNIX 11.23
系统软件	Oracle 10.2.0.3 For HP-UX Itamium

压力测试服务器为一台 IBM PC 服务器，其硬件配置如表 9-11 所示。

表 9-11 测试服务器硬件配置

CPU	Intel Core 2 Duo 2G
内存	2GB
硬盘空间	80GB
IP Address	10.199.2.215

6. 测试脚本与测试数据

测试脚本是执行测试场景所要执行的详细操作。

测试数据是基于从 MMIS 系统提取的数据并上传至压力测试数据库。

须在数据库中准备好以下数据。

(1) Equipment Class；

(2) Equipment Class Group；

(3) Equipment Structure；

(4) Location；

(5) Organization；

(6) Work Group；

(7) Equipment；

(8) Work Nature；

(9) Symptom Code，Failure Code，Failure Cause Code，Component Code；

(10) Equipment Structure Definition；

(11) Standard Job Priority；

(12) BOL/ BOM；

(13) Resource Type；

(14) Stock Code；

(15) Work Order。

7. 压力测试流程

压力测试流程的相关步骤依次如下。

(1) 分析及提取 MMIS 系统的典型业务。

(2) 记录脚本，设定相关参数，如压力序列、压力方法等。

(3) 设置监控参数。

(4) 运行脚本并收集图表。

(5) 继续运行数次,汇总各次测试的测试结果。

(6) 分析测试结果并确定系统的瓶颈。

(7) 制定优化方案,确定问题是否来自硬件或软件。

8. 测试结果分析

下面简要地说明上述测试场景的测试结果。

对于所有场景,每 3s 会添加一个在线用户。在执行脚本之前,会设定 5s 的延迟时间。场景将在 30min 内完成测试。

50 个在线用户同时在不同模块执行各种操作,如表 9-6 所示。针对各个测试场景,按照以下标准进行测试结果分析。

1) 应用性能

(1) 关注应用响应时间;

(2) 各项动作的平均处理响应时间。

2) 数据库性能

3) 系统性能

(1) 应用服务器利用率;

(2) 数据库服务器利用率。

4) 结论

运行总结如表 9-12 所示。

表 9-12 运行总结

已执行测试	MMIS_Customize_Sch
活跃用户	0
已完成用户	50
总用户	50
共用时间/H:M:S	0:30:01
运行状态	完成
所有页面平均响应时间/ms	568.74

性能总结如表 9-13 所示。

表 9-13 性能总结

事件名称	响应时间/ms-平均	响应时间/ms-标准差
Add Association For StandardJob	103.63	30.28
Add BOL For StandardJobParameter	43.87	15.7
Add BOM For StandardJobParameter	174.5	387.03
Add EquipmentRegistration	51.53	30.85
Add Fitment/Refitment History	18.07	5.44
Add Incident	343.72	21.76
Add Location	56.39	202.17

续表

事件名称	响应时间/ms-平均	响应时间/ms-标准差
Add MeasurementReading	39.17	16.5
Add MovementHistory	19.86	10.32
Add Opportunity For StandardJob	94.43	44.85
Add OtherCotst For StandardJobParameter	507.2	19.17
Add PM Master	227.12	327.74
Add Requirement For StandardJob	206.73	478.12
Add Scheduling For StandardJobParameter	29.67	19.19
Add StandardJob	413.7	13.47
Add StandardJobPattern	16 252	4422.49
Add Superseding For StandardJob	694.33	672.41
Add Tasks For StandardJob	104.4	323.76
Add TimeSheet	74.27	202.63
Add Work Order By Full Form	6803.14	270.03
Add Work Order By Quick Form	1266	443.69
Add WorkOrder By Request	795	35.73
Add WorkRequest	3464.51	86.97
Calculate MeasurementReading	166.33	377.6
Change Work Order Status	50.21	50.59
Click Planning Workbench Filter	304.98	226.23
Find Work Order By Full Form	27.44	11
Find Work Order By QuickForm	18.37	7.96
Input Planning Workbench Date Range	1421.3	67.66
Login MMIS	30.15	23.05
Login MMIS By Recipient01	31.5	0.71
Open & Reset PM Master Page	700.57	16.74
Open & Reset PM Master Search Page	245.78	31.81
Open Change Work Order Page	33.25	13.82
Open Equipment Registration Page	1838.94	430.62
Open Fitment/Defiment History Page	18	10.46
Open Incident Page For Add Incident And Update Incident	453.41	136.52
Open Location Page	37.15	164.31
Open MMIS Home Page	3852.04	381.33
Open MMIS Home Page By Recipient01	3750	0
Open MMIS Login Page	222.94	73.72
Open MMIS Login Page By Recipient01	227	55.15
Open Planning Workbench Filter Page (1)	245.54	25.22
Open Planning Workbench Filter Page (2)	18.65	32.62
Open Planning Workbench Follow Up Tag	22.65	8.59
Open Planning Workbench Planned Tag And Caculate Planning Workbench	127.15	334.78
Open PM Master Search Page	1818.68	223.33
Open PM Master Search Result Page	697.79	267.24

续表

事 件 名 称	响应时间/ms-平均	响应时间/ms-标准差
Open Search Measurement Statistics Page	39.17	74.01
Open StandardJobParameter Page	16.83	12.37
Open Work Order Page For Find Work Order By QuickForm	14.85	7.98
Recalculate All Statistics	18.33	6.22
Recalculate Statistics	15.67	9.81
Reset Planning Workbench Filter	89.72	256.72
Search PM Master	770.88	19.68
Search Potential PM Master	1542.32	793.75
Search Statistics	341.17	116.59
Tabulate PM Master	47.98	52.26
Update Add Fitment/Refitment History	20.04	37.57
Update EquipmentRegistration	28.61	24.22
Update Incident	30.61	58.13
Update Location	74.56	324.99
Update MeasurementReading	20.5	12.74
Update MovementHistory	19.61	22.81
Update Planning Workbench For Follow Up	53.33	185.85
Update Planning Workbench For Planned	7983.31	44.09
Update PM Master	32.62	15.57
Update StandardJob	82.5	42.61
Update StandardJobParameter	36.41	18.24
Update StandardJobPattern	19.38	7.19
Update TimeSheet	21.36	10.81
Update Work Order By Full Form	29.08	10.45
Update Work Order By Quick Form	20.19	10.73
Update WorkRequest And Send To Recipient	24 114.03	19.1

50 个用户在线时响应时间最高的 10 项事件如图 9-90 所示。

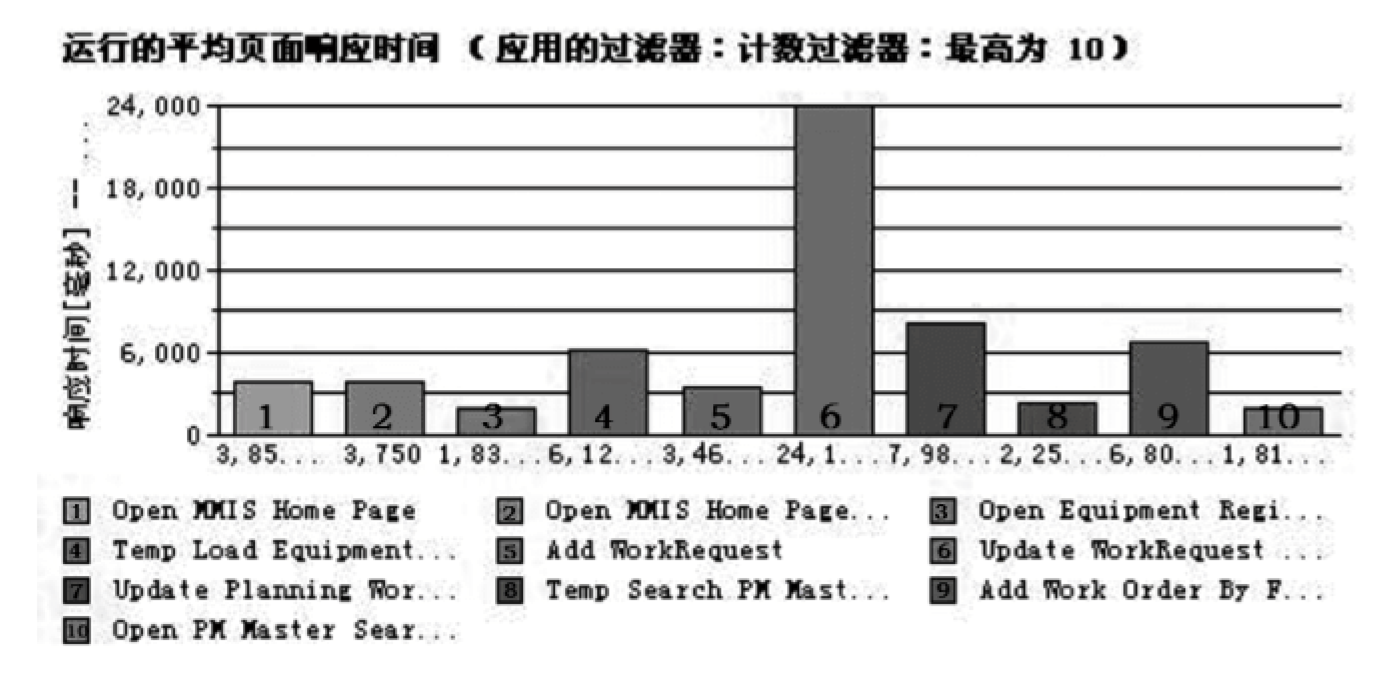


图 9-90　50 个用户在线时响应时间最高的 10 项事件

在 50 个并发用户的场景下，有更多的在线功能所用时间少于预期的目标响应时间。因此得到的结论是，压力测试中采用的硬件配置能支持大于 50 个并发用户。

通过应用服务器的资源情况，得到的结论是，应用服务器的内存能有效改善系统的性能。因此在升级硬件配置的时候应关注内存的容量。

思考题

1. 简要介绍性能测试工具软件 LoadRunner。
2. 通过实例练习使用性能测试工具软件 LoadRunner。
3. 简要介绍安全测试工具软件 AppScan。
4. 通过实例练习使用安全测试工具软件 AppScan。
5. 对比分析 IBM RPT 和 HP LR 这两种性能/压力测试工具。

第10章 测试管理工具

本章学习重点

- 熟悉常见的缺陷管理工具软件 Mantis 的安装与应用。
- 熟悉常见的综合管理工具软件 TestDirector 的安装与应用。

本章学习难点

- 熟悉常见的缺陷管理工具软件 Mantis 的应用。
- 熟悉常见的综合管理工具软件 TestDirector 的应用。

10.1 缺陷管理工具

10.1.1 关于 Mantis

Mantis 是一个基于 PHP 技术的轻量级的开源缺陷跟踪系统，以 Web 操作的形式提供项目管理及缺陷跟踪服务。在功能上、实用性上足以满足中小型项目的管理及跟踪。更重要的是其开源，因此对于企业来说，不需要负担任何费用。

Mantis 是一个缺陷跟踪系统，具有多种特性，包括：易于安装，易于操作，基于 Web，支持任何可运行 PHP 的平台(Windows，Linux，Mac，Solaris，AS400/i5 等)，已经被翻译成 68 种语言，支持多个项目，为每一个项目设置不同的用户访问级别，跟踪缺陷变更历史，定制我的视图页面，提供全文搜索功能，内置报表生成功能(包括图形报表)，通过 E-mail 报告缺陷，用户可以监视特殊的 Bug，附件可以保存在 Web 服务器上或数据库中(还可以备份到 FTP 服务器上)，自定义缺陷处理工作流，支持输出格包括 csv、Microsoft Excel、Microsoft Word，集成源代码控制(SVN 与 CVS)，集成 Wiki 知识库与聊天工具(可选/可不选)，支持多种数据库(MySQL、MSSQL、PostgreSQL、Oracle、DB2)，提供 WebService(SOAP)接口，提供 Wap 访问。

Mantis 具有以下特点。

(1) 个人可定制的 E-mail 通知功能，每个用户可根据自身的工作特点只订阅相关缺陷

状态邮件；

（2）支持多项目、多语言；

（3）权限设置灵活，不同角色有不同权限，每个项目可设为公开或私有状态，每个缺陷可设为公开或私有状态，每个缺陷可以在不同项目间移动；

（4）主页可发布项目相关新闻，方便信息传播；

（5）具有方便的缺陷关联功能，除重复缺陷外，每个缺陷都可以链接到其他相关缺陷；

（6）缺陷报告可打印或输出为 CSV 格式，1.1.7 版：支持可定制的报表输出，可定制用户输入域；

（7）有各种缺陷趋势图和柱状图，为项目状态分析提供依据，如果不能满足要求，可以把数据输出到 Excel 中进一步分析；

（8）流程定制方便且符合标准，满足一般的缺陷跟踪。

10.1.2 使用 Mantis

1. 登录 Mantis

在登录后进入 Mantis 的主界面，如图 10-1 所示。



图 10-1　Mantis 的主界面

在主界面中可以看到一条工具栏，这就是用户能够使用的所有功能了。在工具栏的下方可以看到有 5 大栏，分别如下。

（1）未指定的：是指问题已经报告，但还没有指定由哪个项目组成员进行跟进的问题列表。

（2）已解决的：指问题已经得到解决，问题的状态为“已经解决”。

（3）我正在监视的：指你正在监视哪些问题，在问题报告中，你被选为监视人。

（4）由我报告的：在这里将会显示由你报告的问题列表。

（5）最近修改：这一栏显示哪些问题报告最近被项目组成员修改了。

2. 问题报告

单击“报告问题”进入以下页面，选择报告的问题所属的项目，如图 10-2 所示。

首页 | 我的视图 | 查看问题 | 报告问题 | 修改日志 | 使用说明 | 个人账号 | 注销　　跳转到该问题编号

选择项目	
选择项目	ds系统
指定预设值	☐
	选择项目

图 10-2　选择所属项目

从下拉框选择完成后，单击“选择项目”按钮，进入问题报告主界面，如图 10-3 所示。

输入报告详细资料	[高级报告]
分类	个人事务模块
出现频率	经常
严重性	次要错误
产品版本	
*摘要	
*说明	
附加信息	

图 10-3　问题报告主界面

在图 10-3 中有些项目是标了红星的，表示这些是必填内容。填好问题报告后，单击“提交报告”按钮，就会将此问题提交到系统，系统将会通过 E-mail 通知项目组的相关人员。

在问题报告的右上角有一个“高级报告”的选项，单击后，将会出现以下页面，如图 10-4 所示。

单击“高级报告”后，会发现比原来多出来的好几个选项，这些都是有利于模拟问题重现的。

3. 问题查询

查找问题，只需单击工具栏上的“查看问题”，就会出现以下界面，如图 10-6 所示。

出现频率	经常
严重性	次要错误
选择平台配置	
或填写	
硬件平台	
操作系统	
版本	
产品版本	
产品构建号	
分派给	

图 10-4　高级报告页面



图 10-5　查看问题界面

在工具栏的下方，有一个深浅色相隔的表格，这些表格的内容正是查找的条件选项，单击其中一选项，就会出现一个下拉框代替任意两个字，让用户输入要查找的内容。输入完成后，单击“筛选”按钮就会将结果返回到下方的表格。

图 10-6 就是返回的查找结果，单击相应的记录可以进行修改。

查看问题 (1 - 2 / 2) [打印报告] [CSV 导出]

		P	编号	#	分类	严重性	状态	最后更新	摘要
☐	✎	^	0000003		基本资料模块	次要错误	新建	04-27-06	test
☐	✎		0000001	1		次要错误	新建	04-26-06	fdsf

☐ 全选 移动 确定

图 10-6　返回查找结果

4. 问题修改

在查找结果的列表上单击“编号”,则会进入问题修改的页面,如图 10-7 所示。

查看问题详细资料(简易) [跳转至注释] [问题提醒]　[>>]　[高级查看] [问题历史] [打印]

编号	分类	严重性	出现频率	报告日期	上次更新
0000003	[ds系统] 基本资料模块	次要错误	经常	04-27-06 11:41	04-28-06 09:49
报告人	administrator	查看状态	公共		
分派给					
优先权	高	完成度	未处理		
状态	新建			产品版本	V1.1
摘要	0000003: test				
说明	2l4324lk324				
附加信息					
问题类型	缺陷				
附件					

修改问题　分派给:　[自身]　将状态改为:　打回　监视问题　创建子项问题　移动问题

图 10-7　问题修改页面

在图 10-7 中一共有 6 个按钮,具体功能如下。

(1) 修改问题:进入问题明细页面进行修改。

(2) 分派给:是指将这个问题分派给哪个人员处理,一般只能选择开发员权限的人员。

(3) 将状态改为:更改问题的状态,将需要输入更改状态的理由。

(4) 监视问题:单击后,所有和这个问题相关的改动都会通过 E-mail 发到监视用户的邮箱。

(5) 创建子项问题:建立一个问题的子项,而这个子项报告的问题是依赖于这个问题而存在的。

(6) 移动问题:将这个问题转移到其他项目中。

图 10-8 是关系、上传文件以及问题注释添加的界面。

⊟关系

新建关联　当前问题　与该问题关联:　添加

是该问题的子项:　0000001　新建　fdsf [删除]

当前问题的部分子项还没有解决或关闭。

⊟上传文件

选择文件 (最大的大小: 2,000k)　浏览...　上传文件

当前没有用户在监视这个问题。

⊟添加问题注释

问题注释

查看状态　私有的

添加问题注释

图 10-8　关系、上传文件与问题注释添加

图 10-9 是问题历史的展示界面。

这个问题没有注释信息

[添加问题注释]

⊟ 问题历史

日期修正	账号	字段	改变
04-27-06 11:41	administrator	新建问题	
04-28-06 09:49	administrator	关联已添加	是该问题的子项： 0000001

图 10-9　问题历史

10.1.3　报表统计

单击工具栏上的“报表统计”，以表格的形式对问题进行统计，有“按问题状态”、“按严重性”、“按项目分类”等，如图 10-10 所示。

统计报表

按问题状态	未解决	已解决	已关闭	合计
新建	2	-	-	2
已关闭	-	-	1	1

按严重性	未解决	已解决	已关闭	合计
次要错误	2	0	1	3

按项目分类	未解决	已解决	已关闭	合计
基本资料模块	1	0	1	2

已解决问题的所用时间(天数)	
时间最长的未解决问题	0000002
最长未解决的时间	0.12
平均时间	0.12
合计时间	0.12

问题状态与开发人员(未解决/已解决/已关闭/合计)	未解决	已解决	已关闭	合计
比	0	0	1	1

按日期	
1	2
2	3
3	3
7	3
30	3
60	3
90	3
180	3
365	3

按完成度	未解决	已解决	已关闭	合计
未处理	2	0	0	2
已修正	0	0	1	1

按优先权	未解决	已解决	已关闭	合计
中	1	0	0	1
高	1	0	1	2

图 10-10　报表统计

10.1.4　Mantis 的管理

本节将介绍 Mantis 如何管理用户、如何管理项目、如何添加自定义字段等一系列功能。

1. 用户管理

单击工具栏中的“管理”，进入默认的界面就是用户管理界面。在这个界面中，能看到一些账户的信息，如一周内注册的用户、从未登录的用户等。界面如图 10-11 所示。

在这个界面中可以完成以下功能。

(1) 删除账号：对于一些从未使用的账号进行删除，可以单击“清理账号”按钮进行删除，可以单击账号列表中的“账号”，进入账号明细进行删除，如图 10-12 所示。

(2) 权限设置：为账号选择一个合适的存取权限的角色，如报告人员、开发人员、管理员等。

(3) 重设密码：单击重设密码，密码将会以 E-mail 的形式发到用户登记的电子邮箱。



图 10-11 用户管理界面



图 10-12 清理账号

（4）添加用户到项目：只有进行过这一操作的用户才有权限对特定项目进行操作。界面如图 10-13 所示。



图 10-13 添加用户到项目

（5）默认账号设置：在这里设定 E-mail 默认的提醒情况还有报告时的默认类型（高级报告还是简单报告），最重要的一点是，“界面语言”一定要设为 Chinese_simplified，否则用户登录时将会显示一个英文界面。

2. 项目管理

在这一栏可以添加一个新项目到 Mantis，方便用 Mantis 管理多个项目的问题。

（1）创建项目，单击“创建新项目”，如图 10-14 所示。

[用户管理][项目管理][自定义字段管理][管理全局配置][管理配置]

所有项目 创建新项目

名称 ▲	状态	启用	查看状态	说明
ds系统	开发中	X	公共	dealer service

图 10-14　创建新项目

（2）按要求输入项目内容，完成后单击“添加项目”按钮，如图 10-15 所示。

[用户管理][项目管理][自定义字段管理][管理全局配置][管理配置]

添加项目	
*项目名称	
状态	开发中
查看状态	公共
上传文件存放路径	
说明	
	添加项目

图 10-15　添加项目

（3）项目配置是在创建项目完成后进行的。在项目管理列表，单击项目名称，进入项目配置界面，如图 10-16～图 10-18 所示。

[用户管理][项目管理][自定义字段管理][管理全局配置][管理配置]

编辑项目	
项目名称	ds系统
状态	开发中
启用	☑
查看状态	公共
上传文件存放路径	
说明	dealer service
	更新项目

删除项目

子项目 新建子项目

添加为子项目

图 10-16　编辑项目

分类

分类	分派给	操作
个人事务模块		编辑 删除
基本资料模块		编辑 删除
资格管理模块		编辑 删除

添加分类

从该项目复制分类　复制分类到该项目



版本

版本	时间戳	操作
V1.1	04-27-2006 10:46 中国标准时间	编辑 删除
V2.0	04-27-2006 10:46 中国标准时间	编辑 删除
V1.0	04-27-2006 10:46 中国标准时间	编辑 删除

添加版本　添加并编辑版本

自定义字段

字段	顺序	操作
问题类型	1 更新	删除

添加这个已存在的自定义字段

这个项目是公共的，所有用户都可以访问该项目。

图 10-17　项目配置

这个项目是公共的，所有用户都可以访问该项目。

添加用户至项目

账号	存取权限	
james test	报告人员	添加用户

从该项目复制用户　复制用户到该项目

管理账号

账号	Email	存取权限	操作
administrator	root@hlb.com	管理员	
bb	bonny@hlb.com	报告人员	删除
james	james@hlb.com	报告人员	
kk	kk@hlb.com	开发人员	删除
test	test@hlb.com	报告人员	

全部删除

图 10-18　添加用户至项目、管理账号

在图 10-16～图 10-18 中共分为以下 5 大块功能。

① 子项目：在这里新建一个子项目并设定关联。

② 分类：设定该项目的模块分类。

③ 版本：设定该项目的版本号。

④ 自定义字段：管理另一栏“自定义字段管理”，添加自定义字段，必须在这里设定为属于该项目，否则页面将不会显示。

⑤ 添加用户至项目：设定哪些用户有权访问该项目的内容。

3. 自定义字段管理

通过该功能，不仅能使用 Mantis 预定好的字段，还可以添加自定义的字段，使项目、问题报告更详尽、更适宜。

(1) 新建：输入自定义字段的名称，然后单击“新建自定义字段”按钮，如图 10-19 所示。

[用户管理][项目管理][自定义字段管理][管理全局配置][管理配置]

自定义字段

名称	项目名称	类型	可能取值	默认值	高级
ff	0	字符串			
问题类型	1	枚举类型	缺陷|aaa	缺陷	
报告类型	0	字符串			

新建自定义字段

图 10-19 自定义字段

(2) 名称添加完成后，系统会提示“操作成功”，单击“继续”按钮，进入如图 10-20 所示界面进行修改。

[用户管理][项目管理][自定义字段管理][管理全局配置][管理配置]

修改自定义字段	
名称	报告类型
类型	字符串
可能取值	
默认值	
正则表达式	
读权限	查看人员
写权限	查看人员
最小长度	0
最大长度	0
高级	☐
在创建问题时显示	☑
在更新问题时显示	

图 10-20 修改自定义字段

(3) 可以设定字段的类型、默认值、预设值、长度、修改权限等。不过需要注意一点，如果要添加下拉框(即枚举类型)，选项内容要以“|”隔开，否则所有选项都会连在一起，并且只有一项选择。

4. 管理全局配置

在这里可以设定测试环境，方便在问题报告时，以选择的方式调出。输入的内容很简

单，根据提示输入就可以了，如图 10-21 所示。

[用户管理][项目管理][自定义字段管理][管理全局配置][管理配置]

添加平台配置	
*硬件平台	
*操作系统	
*版本	
简要说明	
* 必填	添加平台配置

编辑或删除平台配置	
	⦿ 编辑平台配置 ○ 删除平台配置
选择平台配置	intel C4 2.0 512ddr win2000 +sql server2000 sp 4
	提交

图 10-21 管理全局配置

5. 管理配置

管理配置分成 4 大块：权限报表、工作流开始、工作流、邮件提醒。

1）权限报表

权限报表列出 Mantis 的 6 大角色所具有的权限，只是一个查询功能，如图 10-22 所示。

新闻

权限	查看人员	报告人员	修改人员	开发人员	经理	管理员
查看私有新闻				√	√	√
管理新闻					√	√

附件

权限	查看人员	报告人员	修改人员	开发人员	经理	管理员
查看附件列表	√	√	√	√	√	√
下载附件	√	√	√	√	√	√
删除附件				√	√	√
上传问题相关附件		√	√	√	√	√

图 10-22 权限报表

2）工作流开始

这一功能实现上就是权限的设定，如图 10-23 所示。

3）工作流

工作流就是指定问题的状态存取权限以及状态的下一状态，如图 10-24 所示。

4）邮件提醒

该功能就是设定当问题发生何种变化，以 E-mail 通知哪些人，如图 10-25 所示。

注意：这些设置只对ds系统项目产生影响。

在下面的表中，将应用下列颜色：
项目设置将覆写其他项目。
所有的项目设置都将覆写默认设置。

问题							
	存取权限级别						
权限	查看人员	报告人员	修改人员	开发人员	经理	管理员	谁可以修改这个值
报告问题	☐	☑	☑	☑	☑	☑	管理员
新建问题的状态设置为	新建						管理员
修改问题	☐	☑	☑	☑	☑	☑	管理员
允许关闭解决状态问题	☐						管理员
允许报告人关闭问题	☐						管理员
监视问题	☐	☑	☑	☑	☑	☑	管理员
管理问题	☐	☐	☐	☑	☑	☑	管理员

图 10-23　工作流权限设定

注意：这些设置只对ds系统项目产生影响。

在下面的表中，将应用下列颜色：
项目设置将覆写其他项目。
所有的项目设置都将覆写默认设置。

影响工作流的起点		
起点	状态	谁可以修改这个值
新建立的问题的状态	新建	管理员
被设置为已解决的问题的状态	已解决	管理员
重新打开的问题的状态	打回	管理员

工作流								
	下一状态							
当前状态	新建	打回	公认	已确认	已分派	已解决	已关闭	默认值
新建		☑	☑	☑	☑	☑	☑	打回
打回	☑		☑	☑	☑	☑	☑	新建
公认	☑	☑		☑	☑	☑	☑	新建
已确认	☑	☑	☑		☑	☑	☑	新建
已分派	☑	☑	☑	☑		☑	☑	新建
已解决	☑	☑（重新打开）	☑	☑	☑		☑	新建
已关闭	☑	☑	☑	☑	☑	☑		新建

图 10-24　工作流状态设定

EMAIL 提醒										
					存取权限级别					
信息	问题报告人	问题管理人	正在监视这个问题的用户	问题注释添加人	查看人员	报告人员	修改人员	开发人员	经理	管理员
当分派问题时发送 Email 提醒	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
当重新打开时发送 Email 提醒	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
删除时发送 Email	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
当添加问题注释时发送 Email 提醒	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
关联改变时发送 Email	☑	☑	☑	☑	☑	☑	☑	☑	☑	☑
状态已修改为 '新建'	☑	☑	☐	☐	☑	☑	☑	☑	☑	☑

图 10-25　E-mail 提醒

10.2 综合管理工具

10.2.1 TestDirector 简介

TestDirector 是 TestDirector Mercury Interactive 公司推出的基于 Web 的测试管理工具，无论是通过 Internet 还是通过 Intranet 都可以以基于 Web 的方式来访问 TestDirector。

应用程序测试是非常复杂的，它需要开发和执行数以千计的测试用例。通常情况下，测试需要多样式的硬件平台、多重的配置(计算机，操作系统，浏览器)和多种的应用程序版本。管理整个测试过程中的各个部分是非常耗时和困难的。

TestDirector 能够让用户系统地控制整个测试过程，并创建整个测试工作流的框架和基础，使整个测试管理过程变得更为简单和有组织。

TestDirector 能够帮助用户维护一个测试工程数据库，并且能够覆盖应用程序功能性的各个方面。在工程中的每一个测试点都对应着一个指定的测试需求。TestDirector 还提供了直观和有效的方式来计划和执行测试集、收集测试结果并分析数据。

TestDirector 还专门提供了一个完善的缺陷跟踪系统，它能够让用户跟踪缺陷从产生到最终解决的全过程。TestDirector 通过与邮件系统相关联，缺陷跟踪的相关信息就可以被整个应用开发组、QA、客户支持，负责信息系统的人员所共享。

TestDirector 提供了与 Mercury Interactive 公司的测试工具(WinRunner，LoadRunner，QuickTest Professional，Astra QuickTest，QuickTest Professional for MySAP. com Windows Client，Astra LoadTest，XRunner，Visual APIand Visual API-XP)、第三方或者自主开发的测试工具、需求和配置管理工具、建模工具的整合功能。TestDirector 能够与这些测试工具很好地无缝链接，为用户提供全套解决方案选择来进行全部自动化的应用测试。

TestDirector 会指导用户进行需求定义、测试计划、测试执行和缺陷跟踪，即整个测试过程的各个阶段。通过整合所有的任务到应用程序测试中来确保客户收到更高质量的产品。

TestDirector 的测试管理包括如下 4 个阶段，如图 10-26 所示。

图 10-26 TestDirector 测试管理的 4 个阶段

(1) 需求定义(Specify Requirements)：分析应用程序并确定测试需求。

(2) 测试计划(Plan Tests)：基于测试需求，建立测试计划。

(3) 测试执行(Execute Tests)：创建测试集(Test Set)并执行测试。

(4) 缺陷跟踪(Track Defects)：报告程序中产生的缺陷并跟踪缺陷修复的全过程。

10.2.2 安装 TestDirector

在安装盘下执行 setup. exe 文件，如图 10-27 所示。



图 10-27 执行 setup.exe

单击 Next 按钮，输入序列号，单击 Next 按钮。

选择支持的数据库服务器类型：MS-SQL Server，Access，Sybase，Oracle，可选一个或多个，单击 Next 按钮，如图 10-28 所示。



图 10-28 选择数据库

输入连接串，这里采用默认的，单击 Next 按钮，如图 10-29 所示。

这里验证用户是否是域用户，请输入域用户账号，单击 Next 按钮，如图 10-30 所示。



图 10-29　连接串

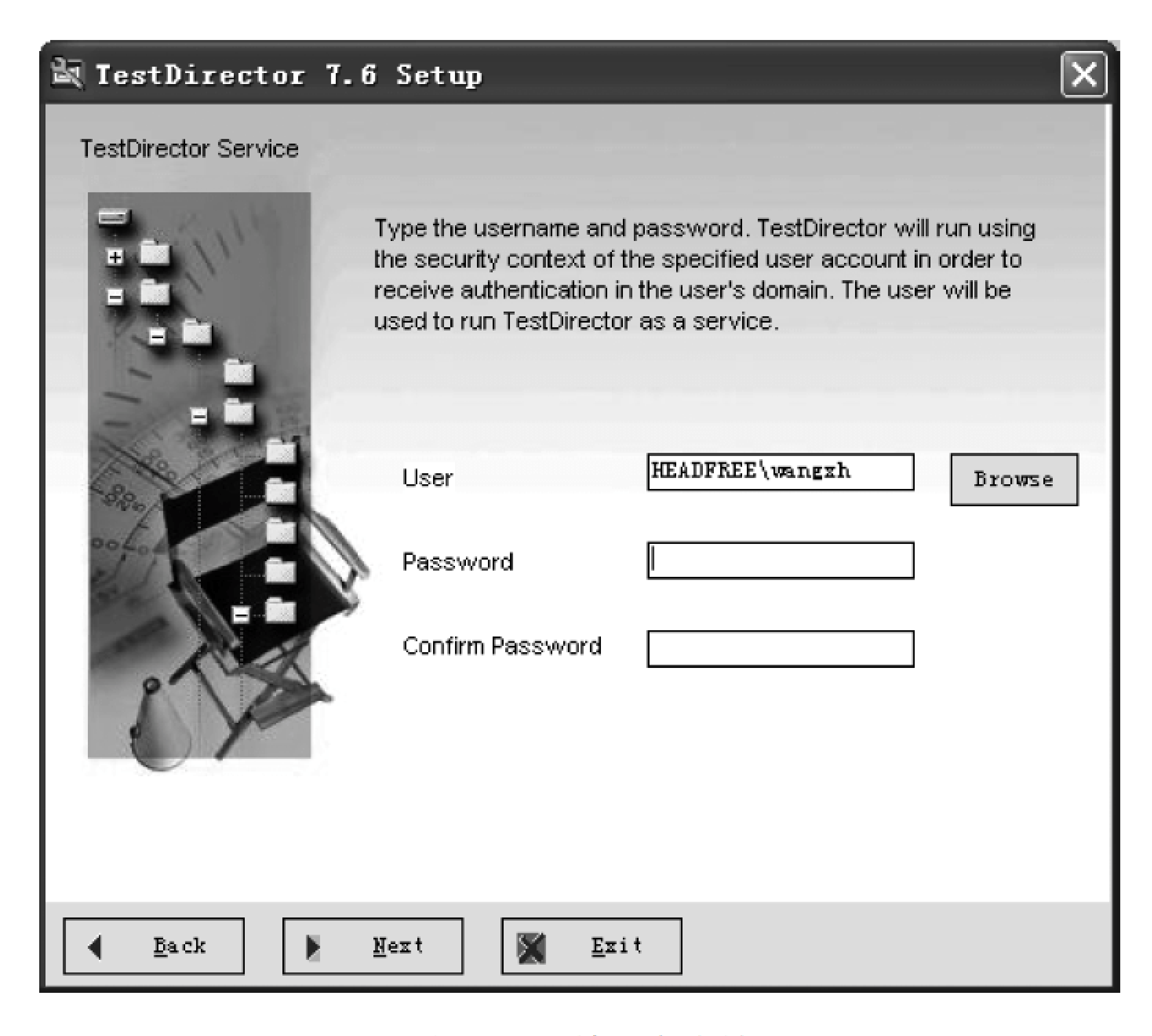


图 10-30　域用户账号

设置域存储区的路径，这里保存了域存储区的相关信息（建议使用默认路径），单击 Next 按钮，如图 10-31 所示。

单击 Yes 按钮，新建这个目录，如图 10-32 所示。

单击 Yes 按钮，共享这个目录，如图 10-33 所示。

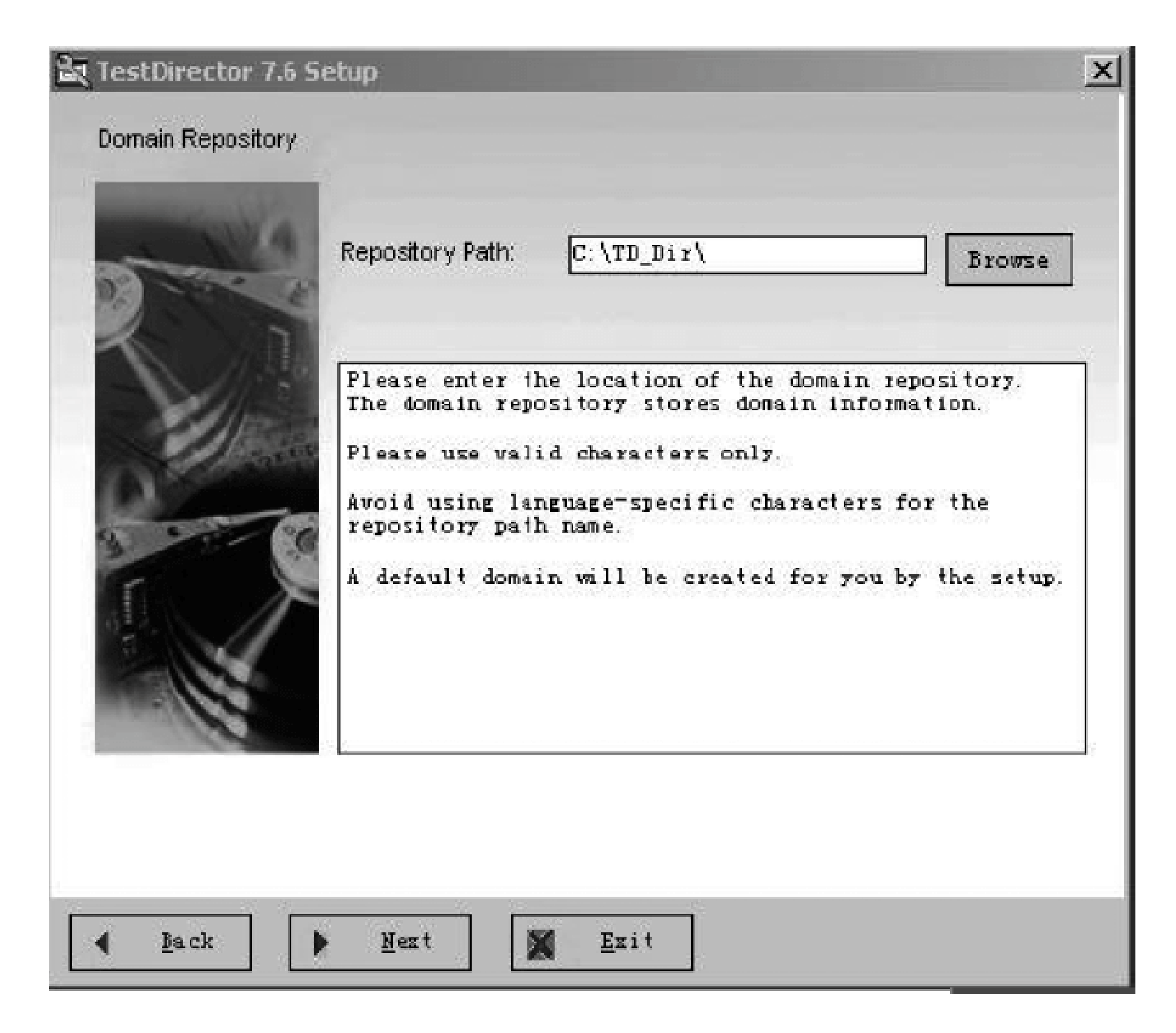


图 10-31 存储路径



图 10-32 确认信息



图 10-33 确定新建目录

设置共享目录名称，单击 OK 按钮，如图 10-34 所示。



图 10-34 共享目录

设置邮件服务：输入用于发送邮件的 SMTP 服务名称(可以是 IP)，或者用系统 IIS 有的 SMTP 服务，单击 Next 按钮，如图 10-35 所示。设置发布 Web 的虚拟目录：输入虚拟目录名称，并指定物理安装位置，使用默认值后单击 Next 按钮。

单击 Yes 按钮，创建这个目录，如图 10-36 所示。

设置是否安装默认的工程，单击 Next 按钮，如图 10-37 所示。

这是一个前面设置的信息摘要，确认正确后，单击 Install 按钮，如图 10-38 所示。

如图 10-39 所示，单击 Finish 按钮，完成 TD 7.6 的安装。重启后生效，如图 10-40 所示。

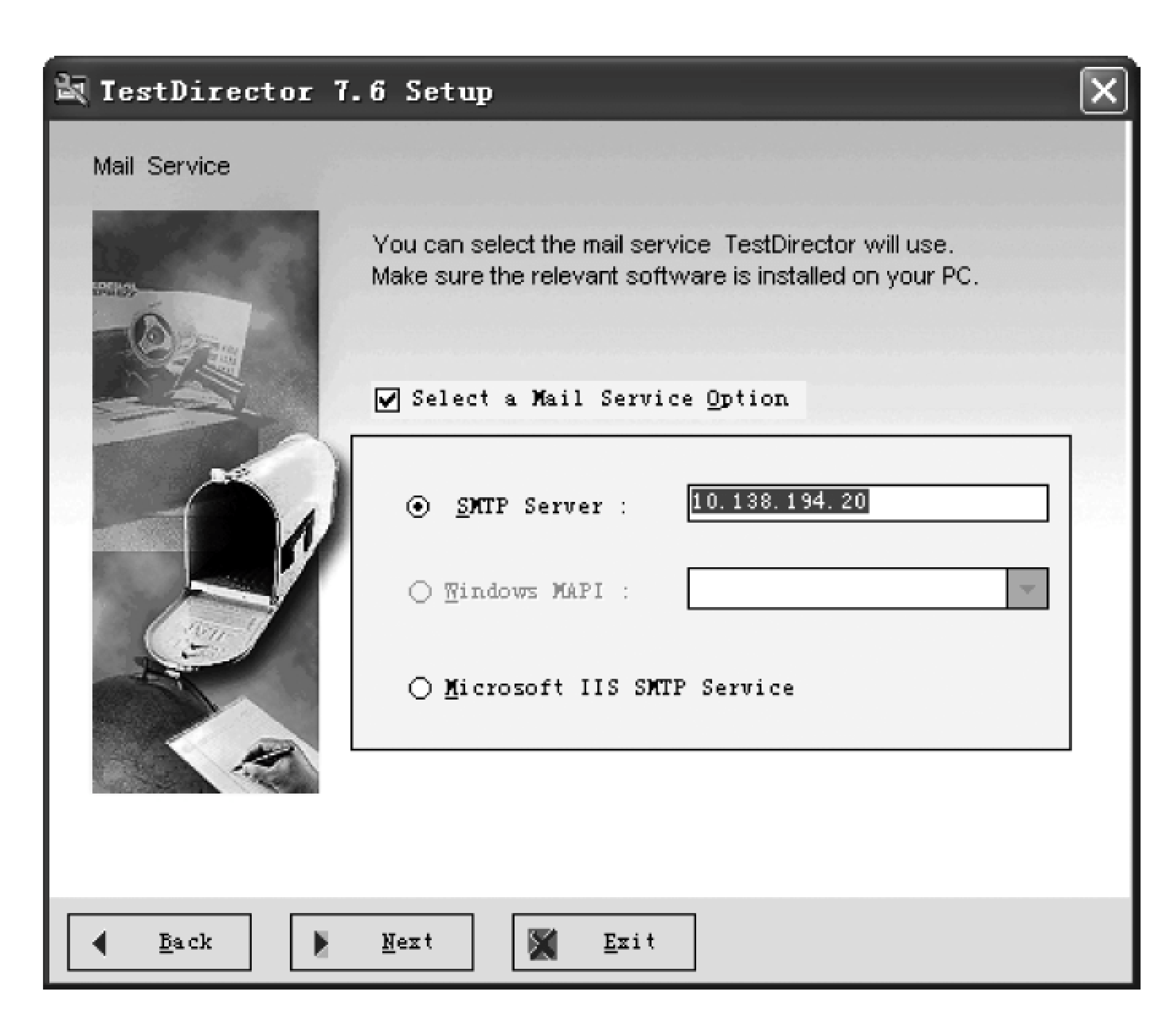


图 10-35　SMTP 服务名称

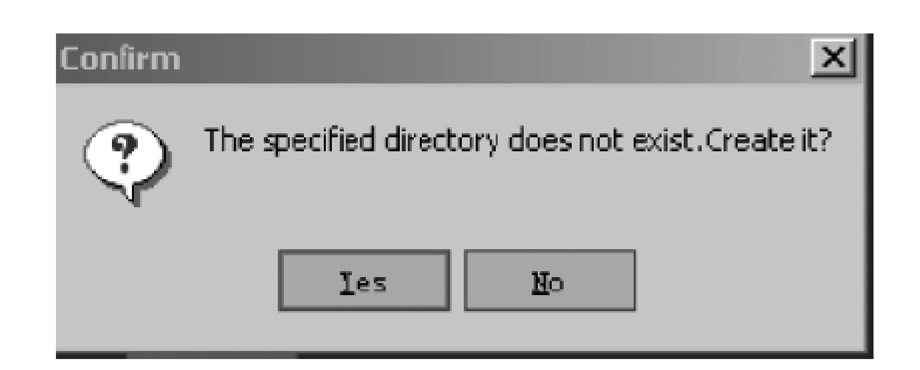


图 10-36　确认信息

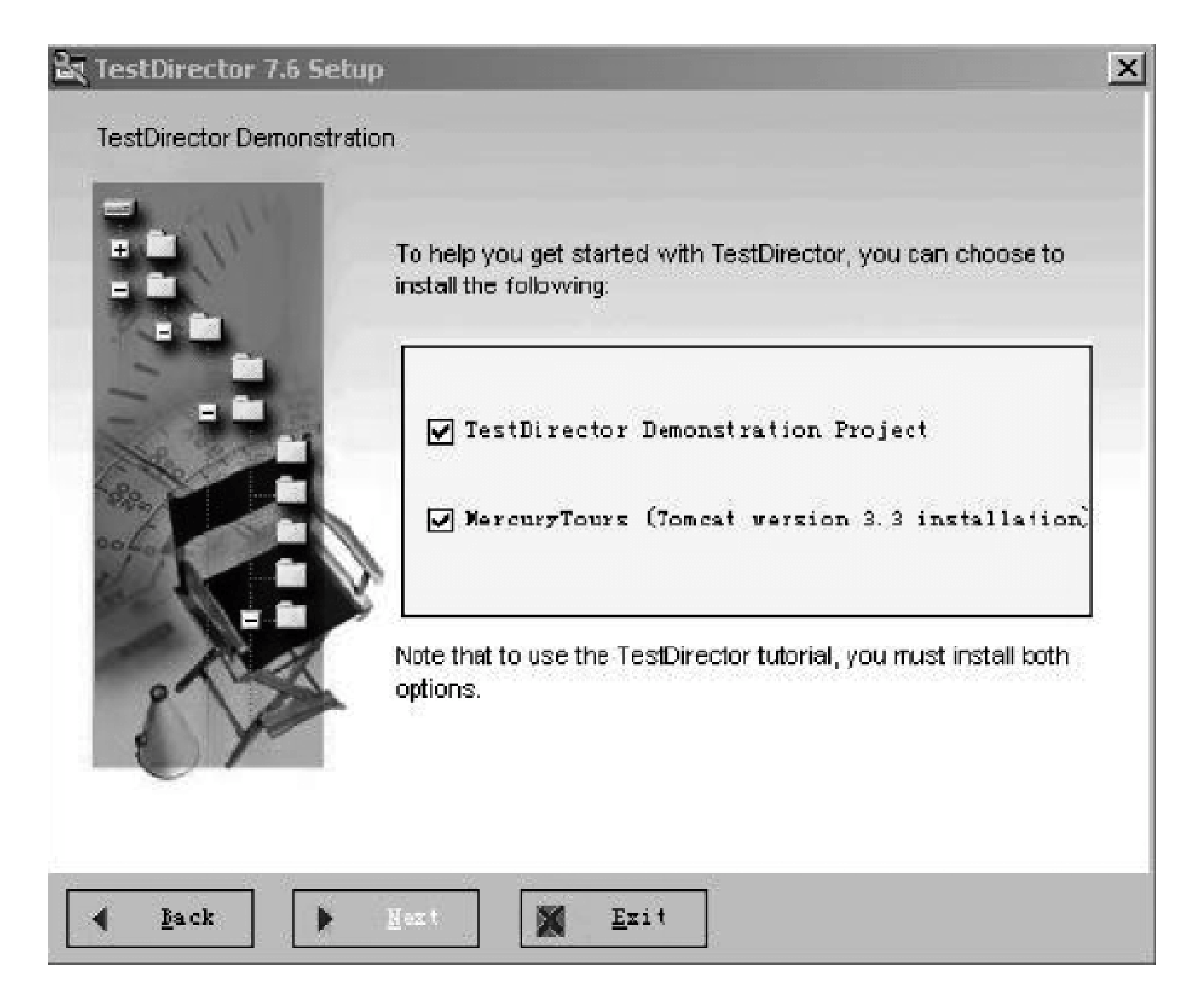


图 10-37　设置是否安装默认的工程

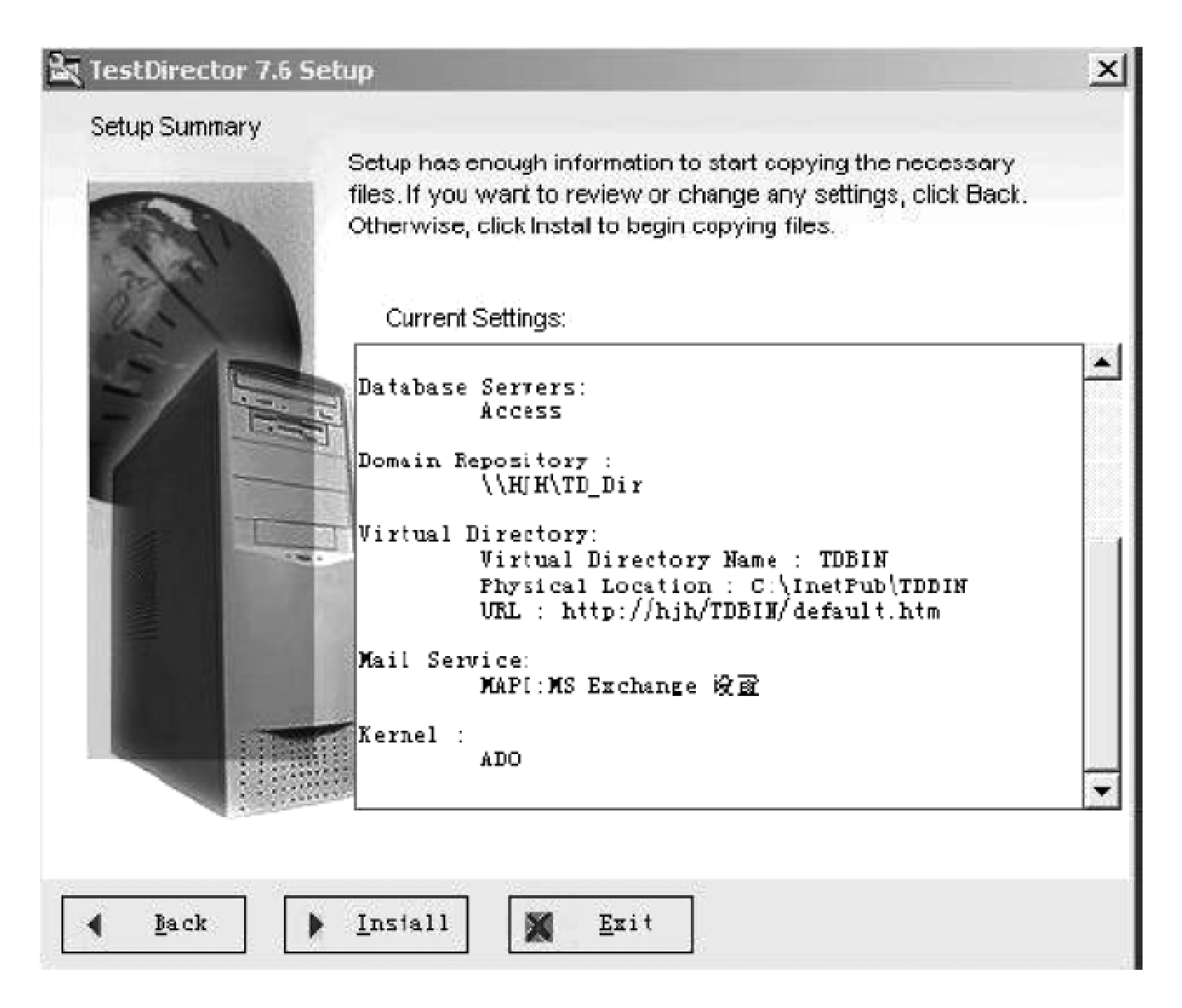


图 10-38 信息摘要



图 10-39 安装完成



图 10-40 重启计算机

10.2.3 创建项目

双击 TestDirector 图标，如图 10-41 所示。

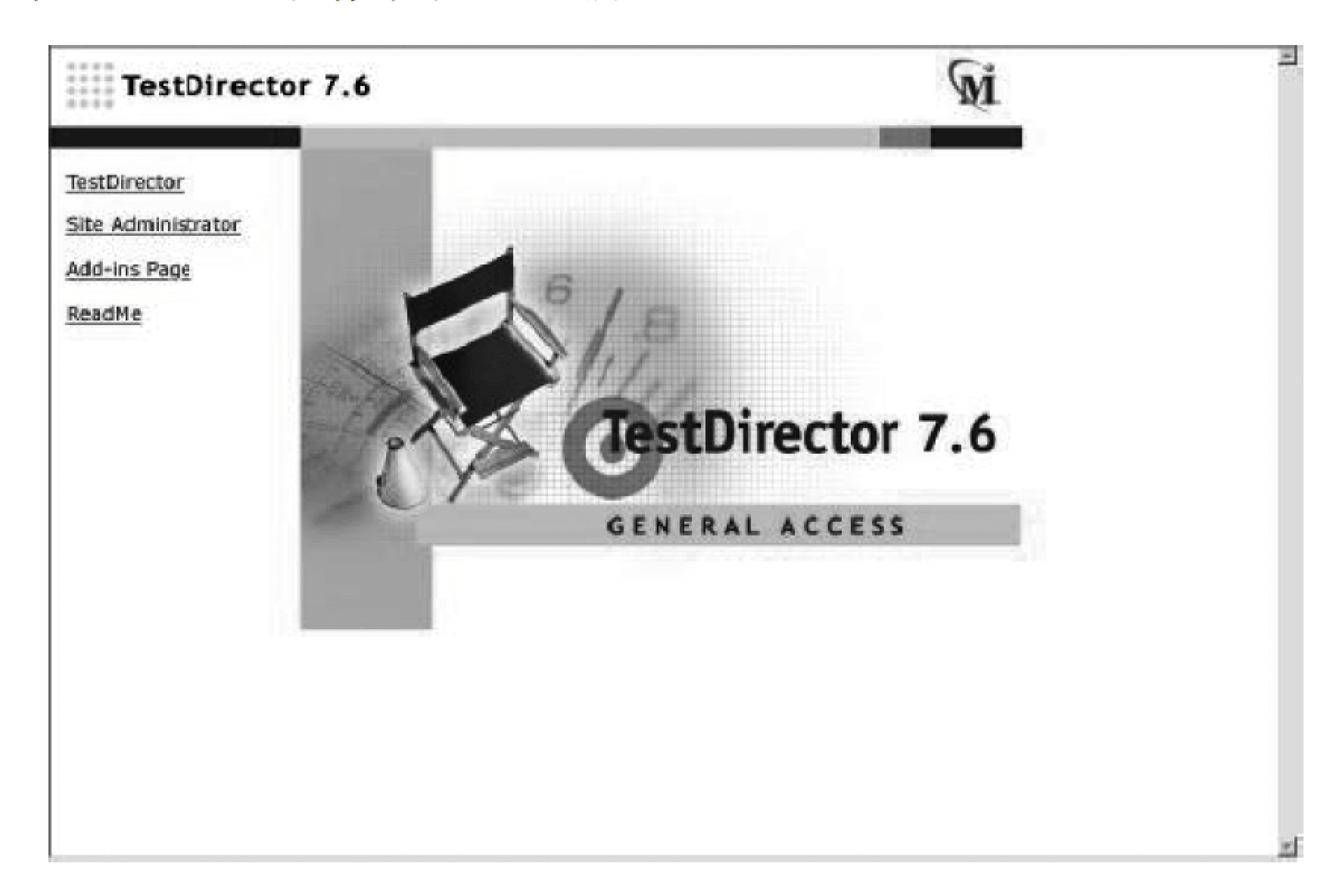


图 10-41 启动 TestDirector

单击 Site Administrator 链接，进入站点管理的登录界面，如图 10-42 所示。



图 10-42 站点管理的登录界面

单击 Login 按钮，进入站点管理的主界面，如图 10-43 所示。

在主界面中单击 Create 按钮，弹出 Greate Project 对话框，如图 10-44 所示。

默认项目名称为以“TD_”为前导字符，可以更改，例如“TD_KM”，当然这个名称可任意输入，但要保证唯一，选择数据类型，如 MS-SQL，单击 Next 按钮。

如图 10-45 所示，输入本地数据库的用户名和密码(若本地服务器的名称不是 TDSQLSERVER，则需要在 SQL Server 的客户端网络使用工具中建立别名 TDSQLSERVER)，单击 Next 按钮。

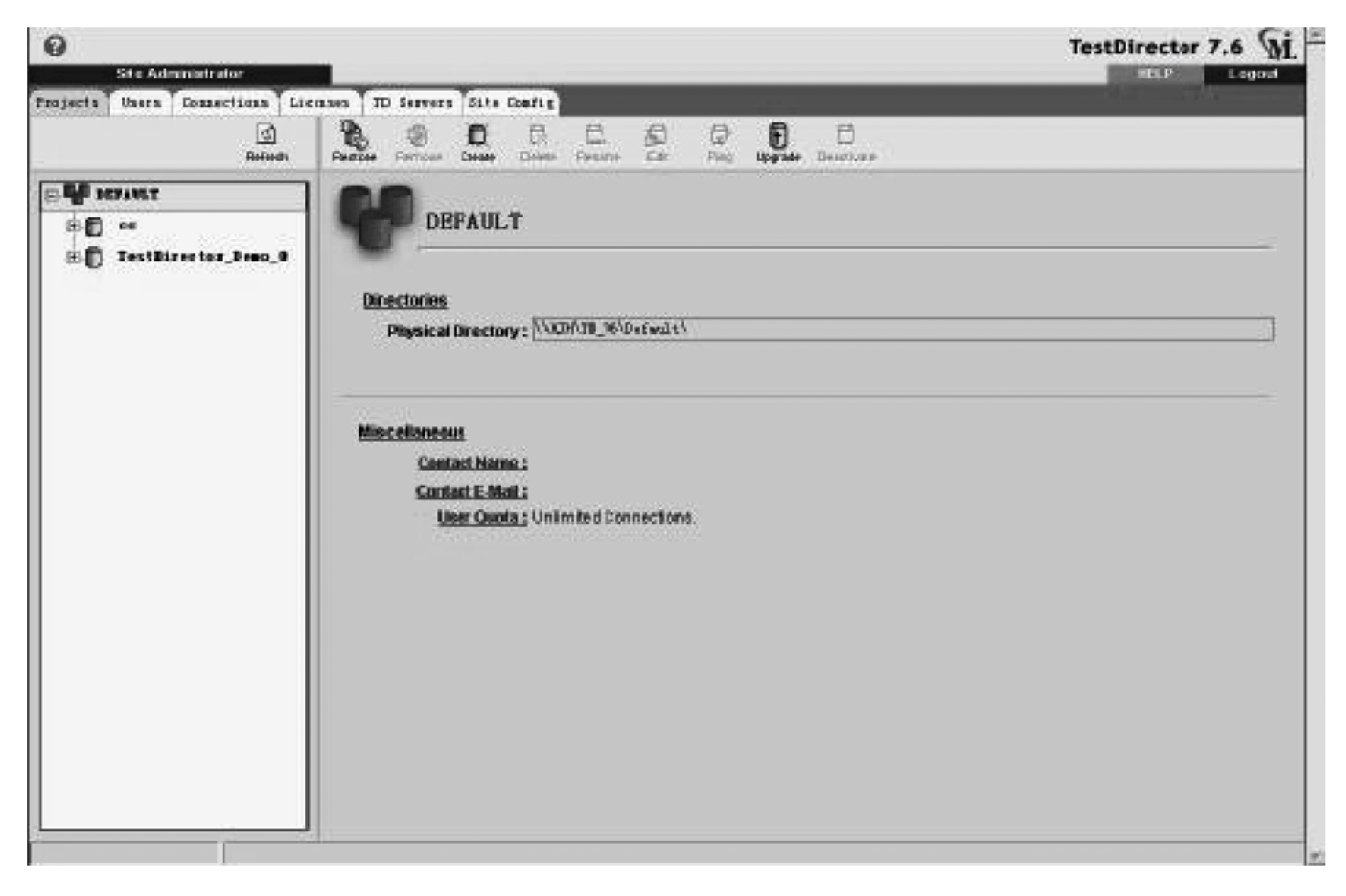
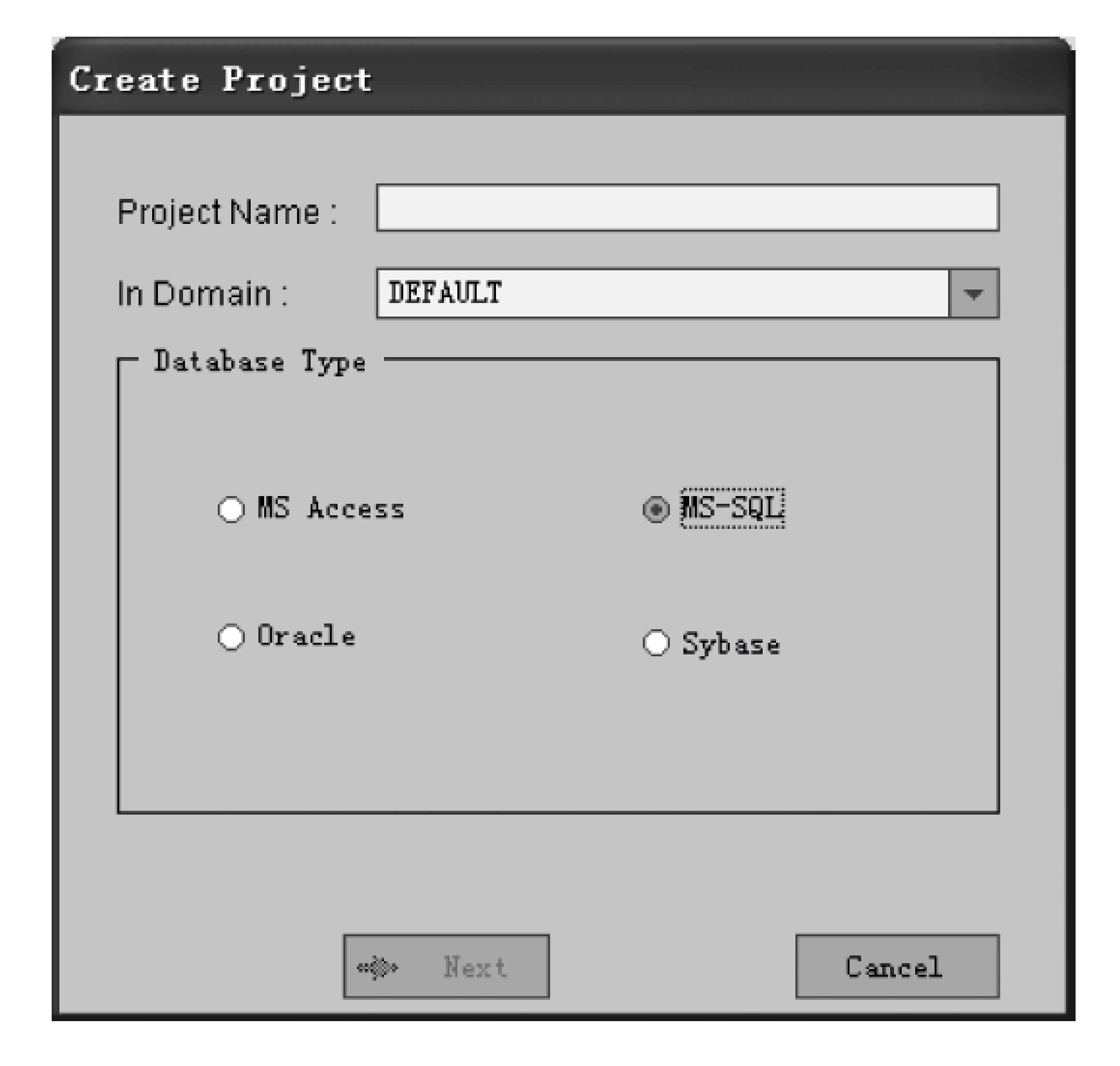


图 10-43 站点管理的主界面



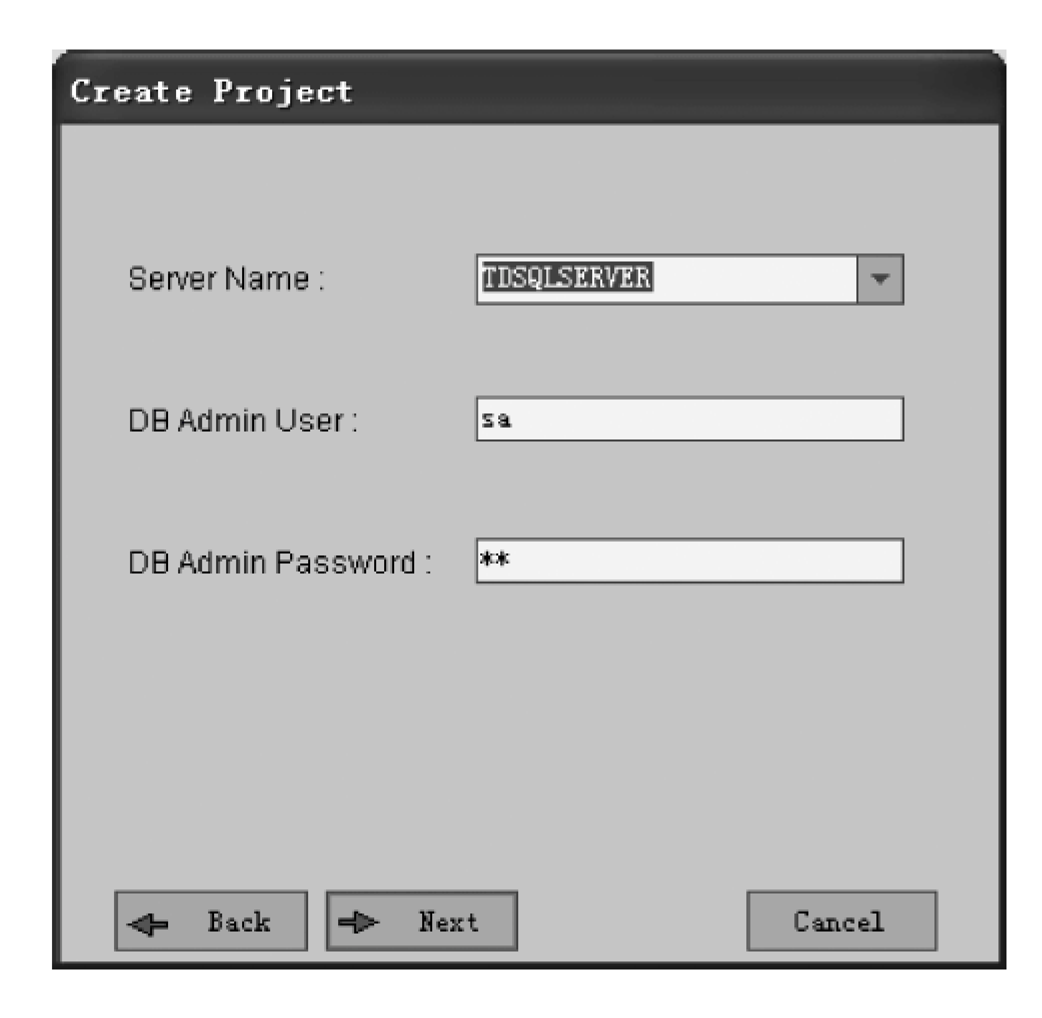



图 10-44 新建项目

图 10-45 输入本地数据库的用户名和密码

单击 Create 按钮就新建了一个项目，如图 10-46 所示。

如果项目是要从已存在的项目复制，那么在新建对话框中单击 Copy 按钮，如图 10-47 所示。

在 Project 选择框中选择要复制的项目，选择要复制哪些类型的数据后单击 Copy 按钮创建。

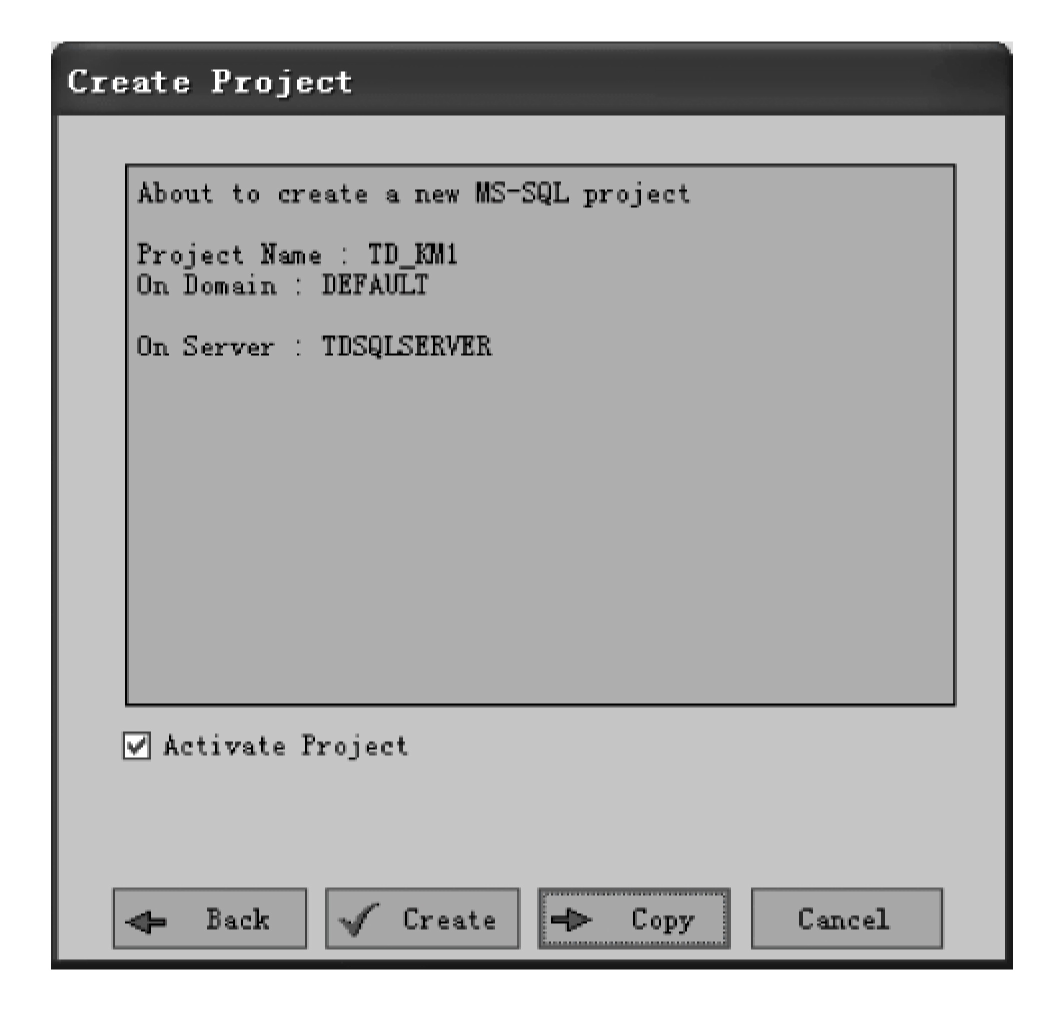


图 10-46　完成新建项目

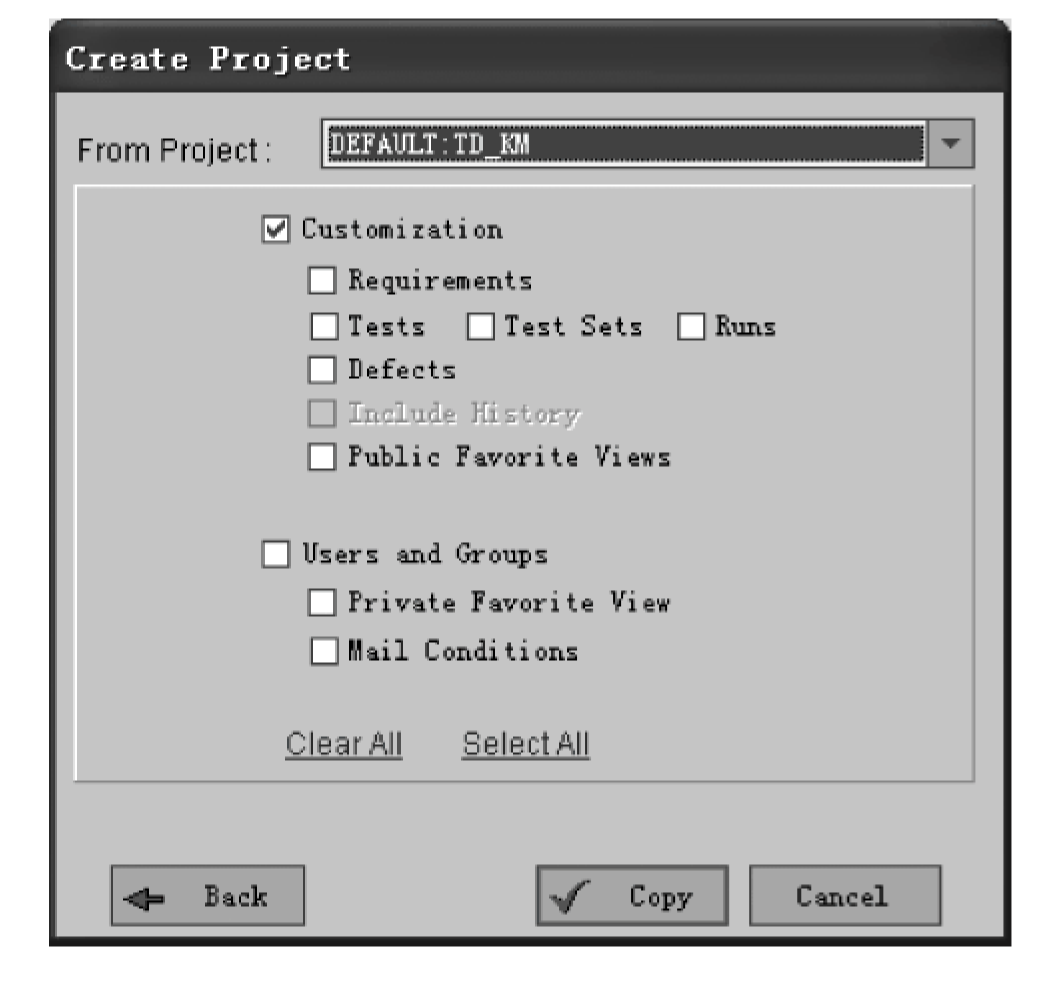


图 10-47　从已存在的项目复制

10.2.4　定制项目模块、加入用户和授权

登录 TestDirector，单击“自定义”按钮，进入工程的界面，如图 10-48 所示。

Login
Domain : DEFAULT
Project : TD_KM
User ID : admin
Password :
OK
Cancel

图 10-48　选择创建的项目

在 Project 中选择创建的项目，输入用户名和密码，密码为空，单击 OK 按钮进入项目定制页面，如图 10-49 所示。

配置工程字段：单击 Customize Project Entities 链接，如图 10-50 所示。

这里有 7 个表，每个表中的字段分为系统字段（System Fields）和用户字段（User Fields），如图 10-51 所示。

选择 DEFECT→User Fields 节点，单击底部的 New Field 按钮，如图 10-52 所示。

输入字段名称和数据类型，并选择显示模式，增加完后，单击 OK 按钮返回。

定制项目的模块分类：单击 Customize Project Lists 进入定制列表框，如图 10-53 所示。

在 Lists 中选择 All Projects，单击 New List 按钮，弹出 New Item 对话框，如图 10-54 所示。

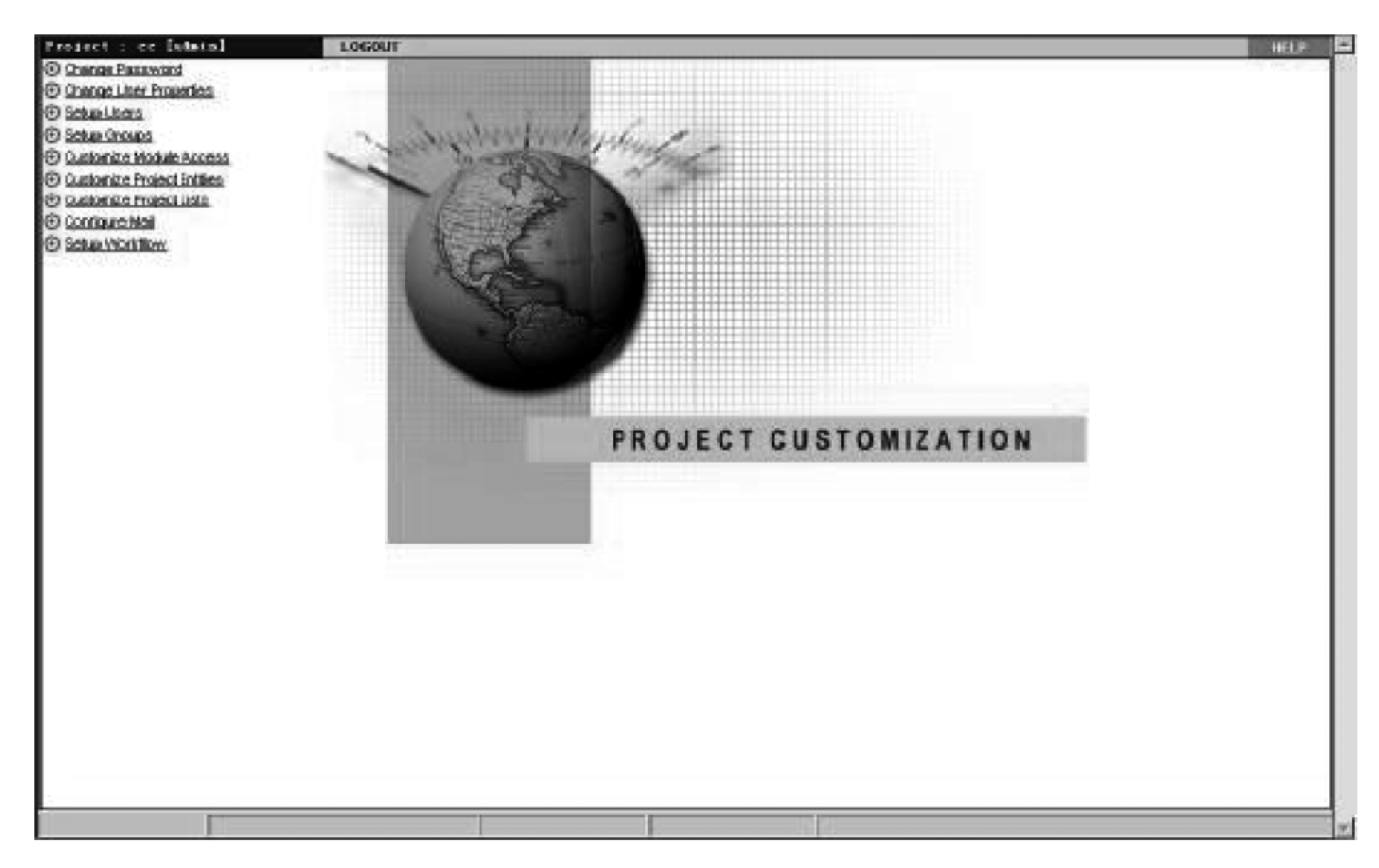


图 10-49 项目定制页面

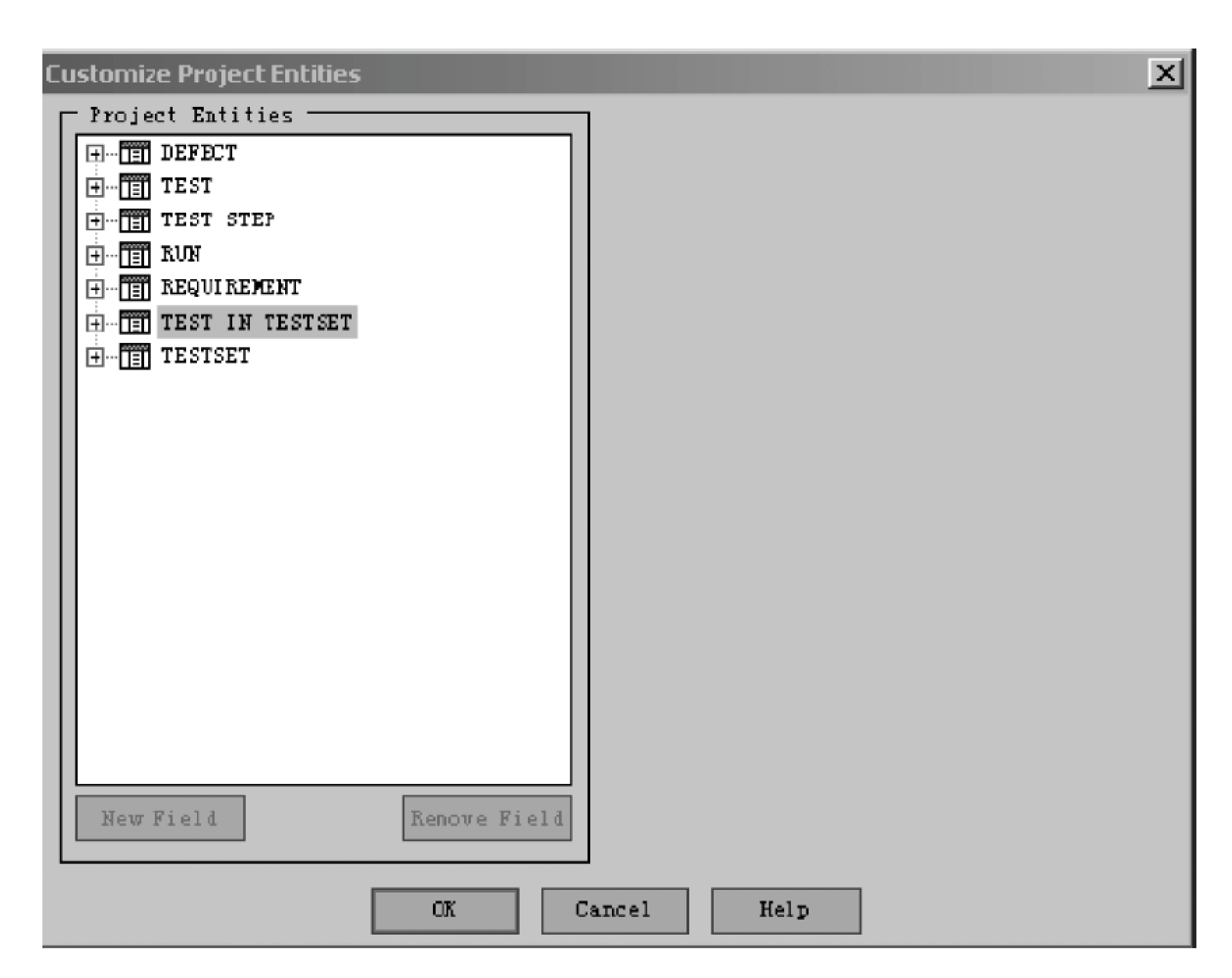


图 10-50 配置工程字段



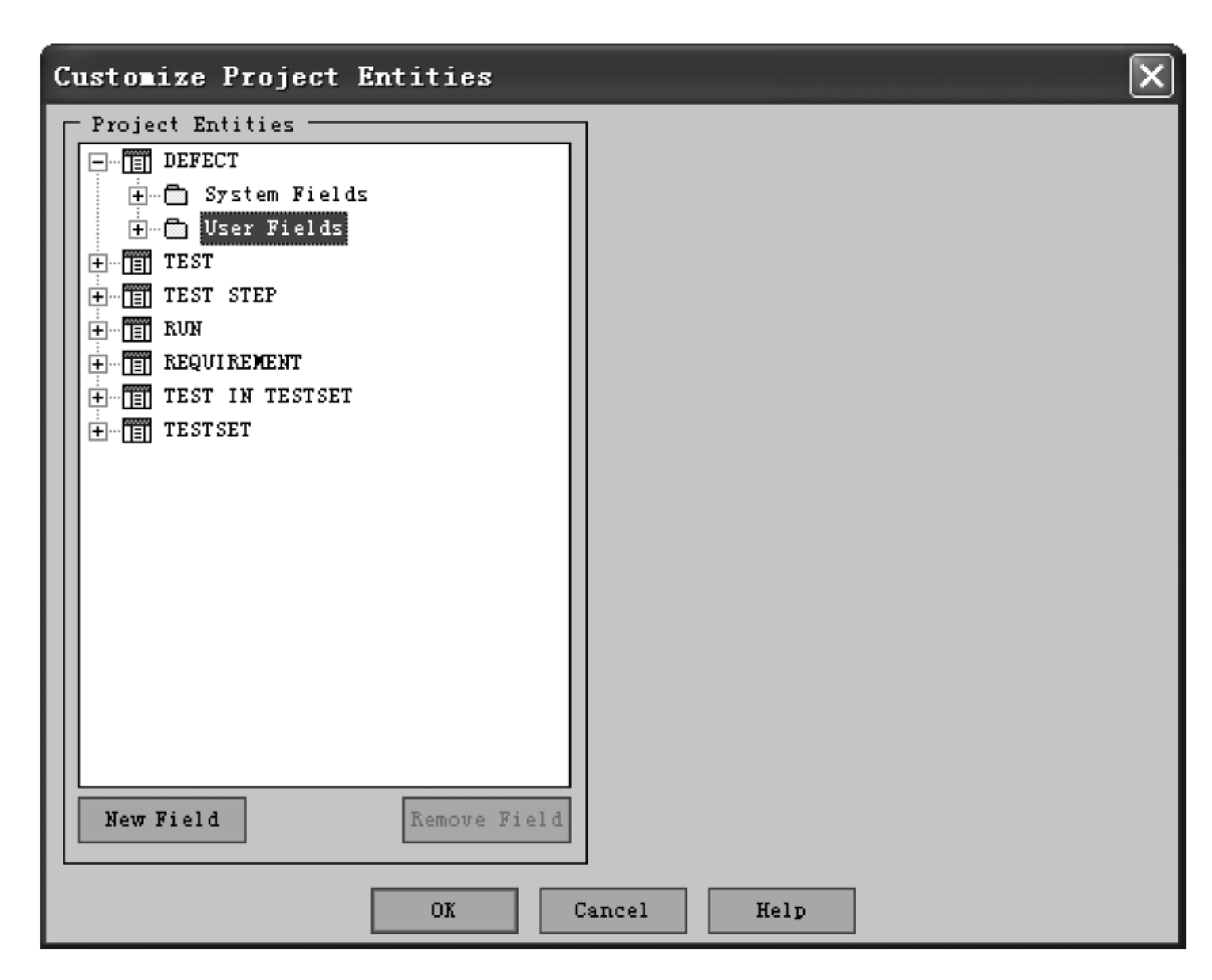


图 10-51　系统字段和用户字段

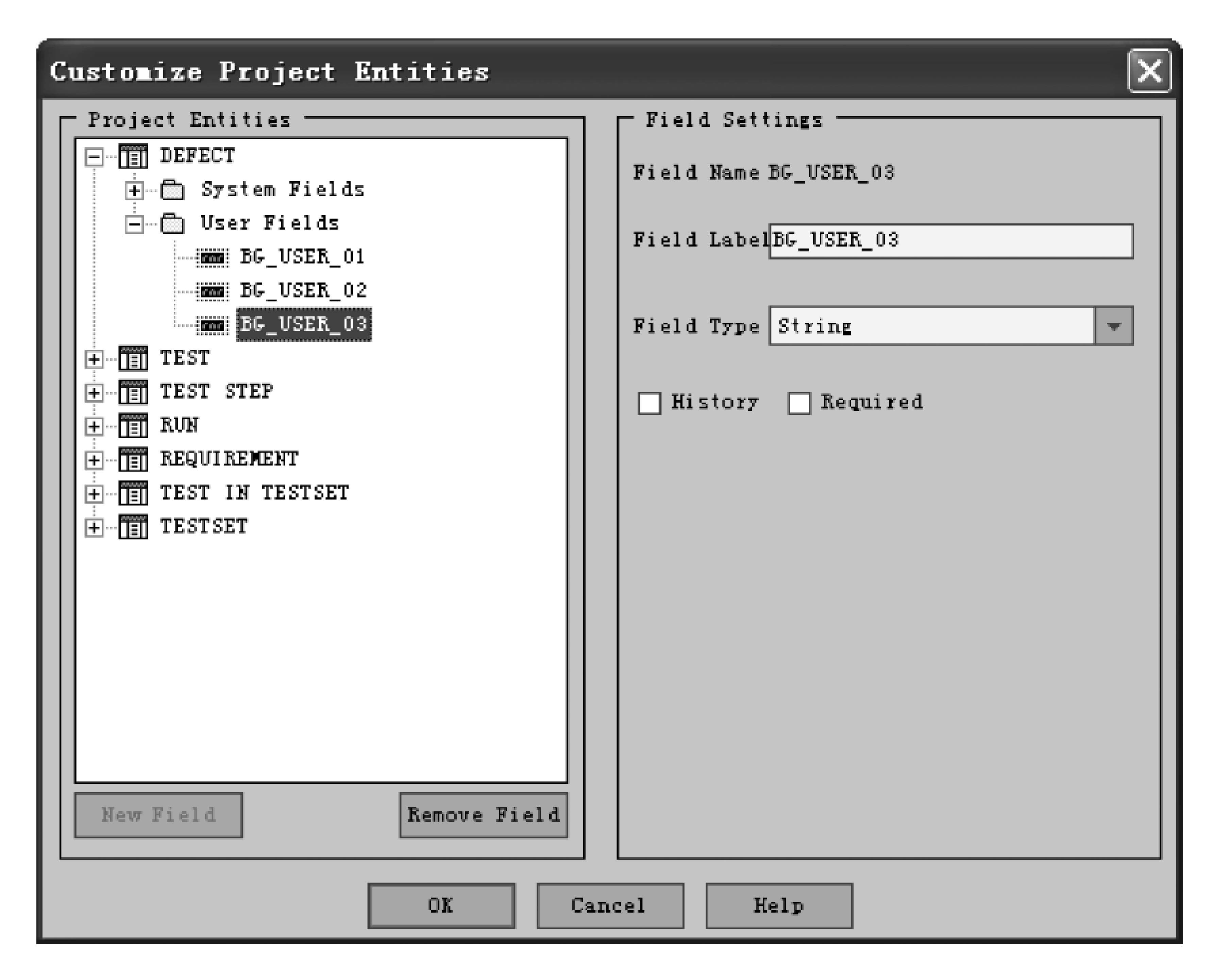


图 10-52　用户字段

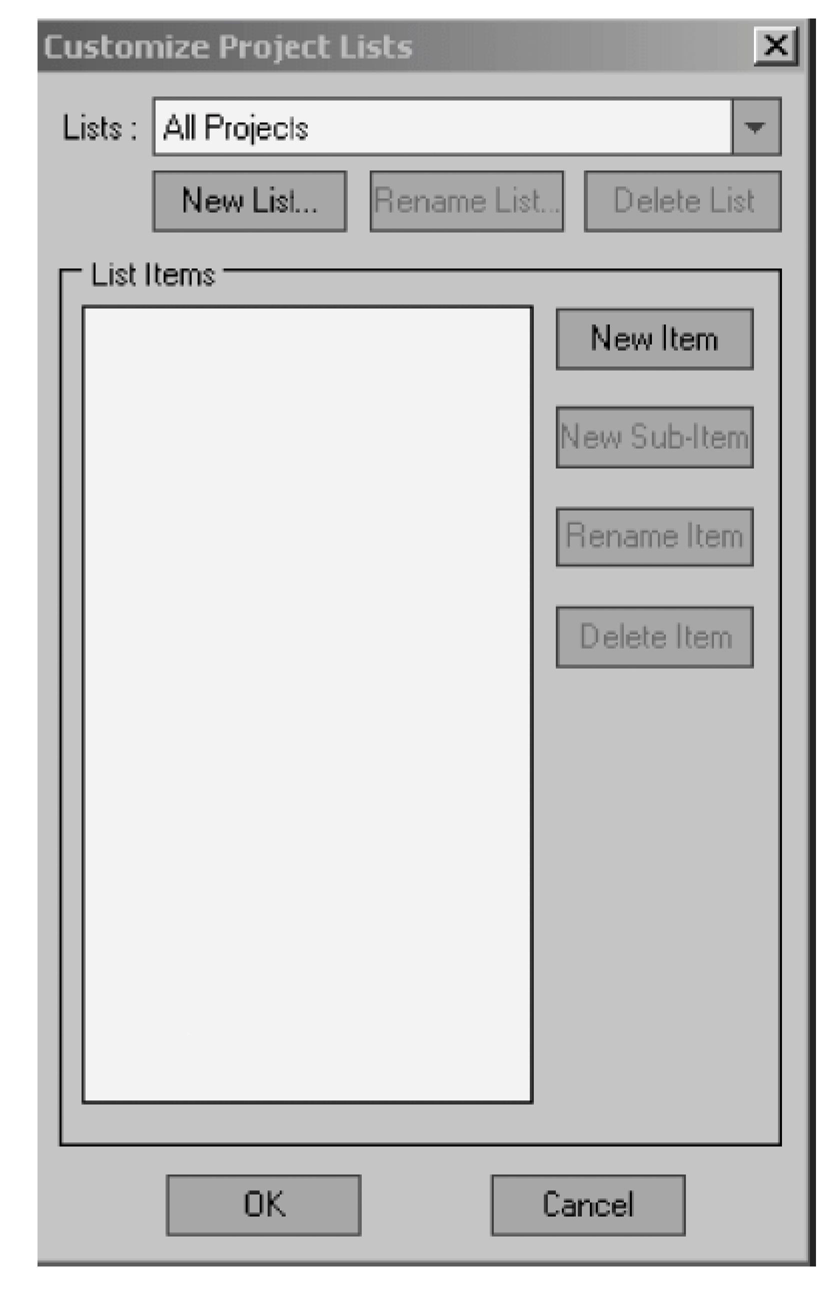


图 10-53 定制列表框



图 10-54 New Item 对话框

输入条目名称，然后单击 OK 按钮返回。

如图 10-55 所示，选择增加进来的条目，单击 New Sub-Item 按钮弹出增加条目对话框。输入条目名称，单击 OK 按钮返回。重复以上操作完成项目各模块的加入，完成后返回定制主界面。

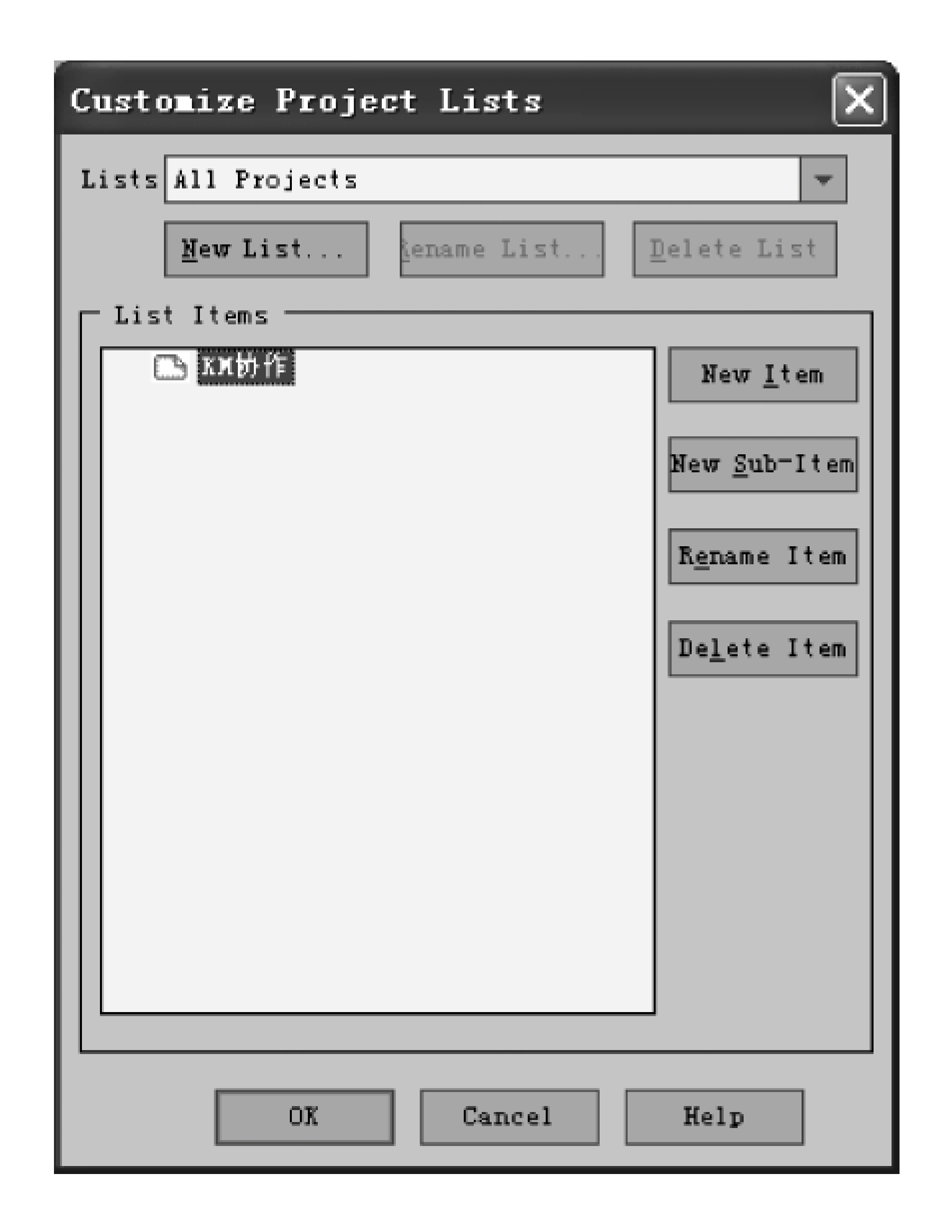


图 10-55 条目对话框

10.2.5　Defect 的使用

打开 TestDirector 链接，进入工程管理的界面。选择项目，输入用户名和密码，登录后进入缺陷管理页面单击工具条上的 Add Defect 按钮，打开 Add Defect 对话框，如图 10-56 和图 10-57 所示。

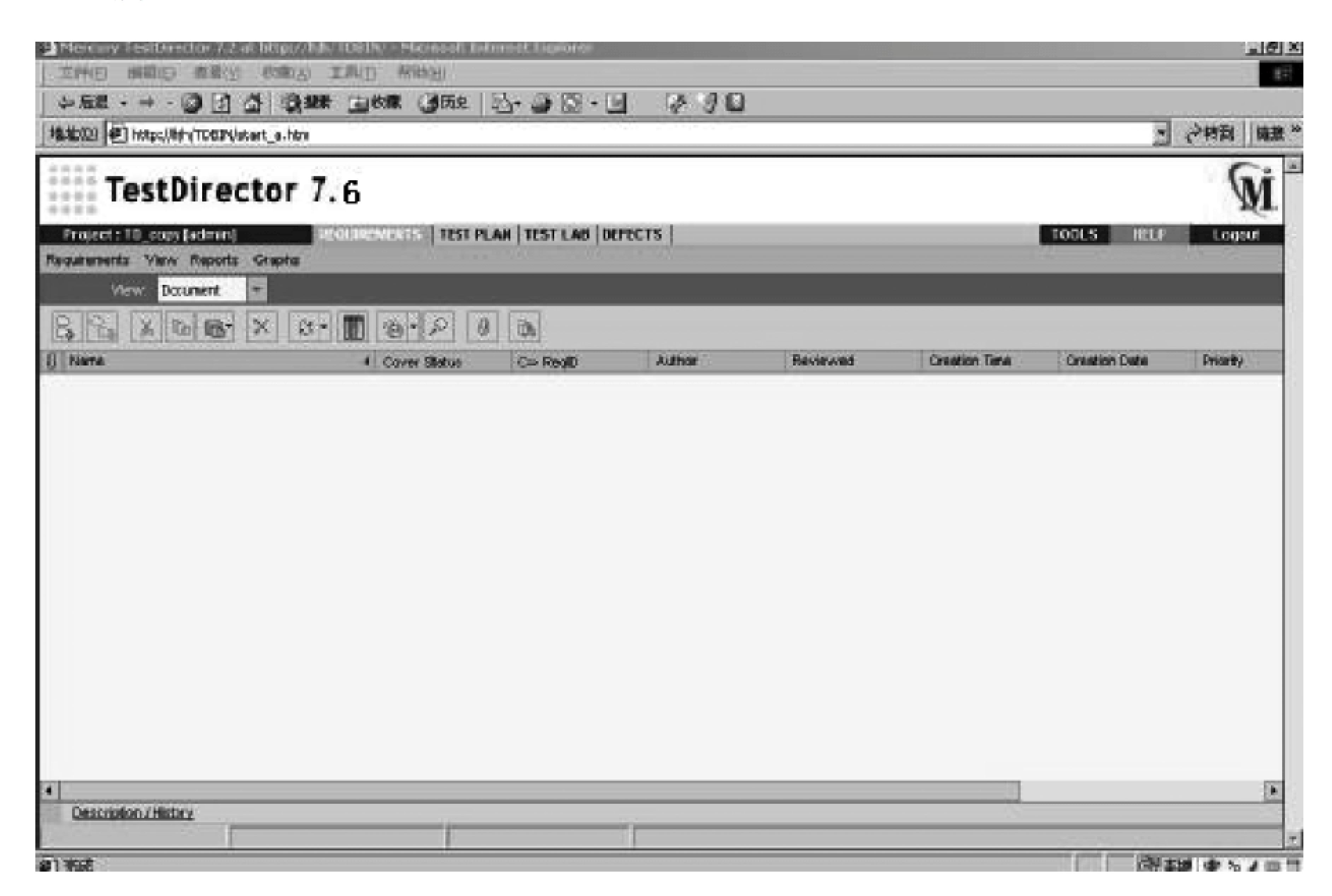


图 10-56　缺陷管理页面

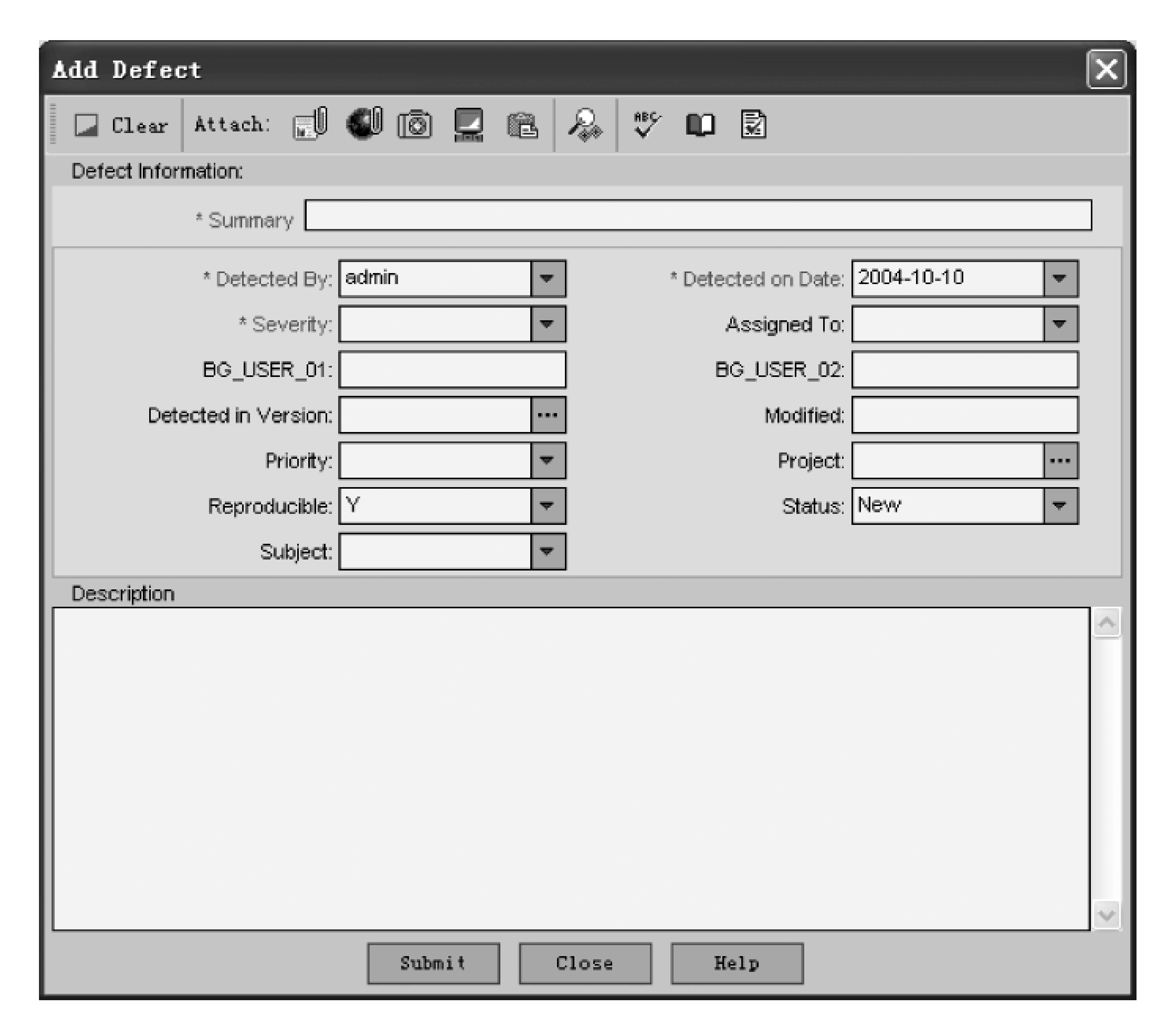


图 10-57　增加缺陷

输入摘要、项目、描述、严重等级、优先等级、标题等信息。顶上有一排按钮，分别是：清除(Clear)、附件(Attach)(依次是文件，Web 页面，屏幕照相，系统信息)、查找相似缺陷(Find similar defects)。

思考题

1. 请简要介绍缺陷管理工具软件 Mantis。它有什么特点？
2. 通过实例练习使用缺陷管理工具软件 Mantis。
3. 请简要介绍综合管理工具软件 TestDirector。
4. TestDirector 的测试管理包括哪几个阶段？
5. 通过实例练习使用综合管理工具软件 TestDirector。



软件测试常用术语表

A.

Acceptance Testing——可接受性测试

一般由用户/客户进行的确认是否可以接受一个产品的验证性测试。

Actual Outcome——实际结果

被测对象在特定的条件下实际产生的结果。

Ad Hoc Testing——随机测试

测试人员通过随机地尝试系统的功能,试图使系统中断。

Algorithm——算法

(1) 一个定义好的有限规则集,用于在有限步骤内解决一个问题;

(2) 执行一个特定任务的任何操作序列。

Algorithm Analysis——算法分析

一个软件的验证确认任务,用于保证选择的算法是正确的、合适的和稳定的,并且满足所有精确性、规模和时间方面的要求。

Alpha Testing——Alpha 测试

由选定的用户进行的产品早期性测试。这个测试一般在可控制的环境下进行。

Analysis——分析

(1) 分解到一些原子部分或基本原则,以便确定整体的特性;

(2) 一个推理的过程,显示一个特定的结果是假设前提的结果;

(3) 一个问题的方法研究,并且问题被分解为一些小的相关单元做进一步详细研究。

Anomaly——异常

在文档或软件操作中观察到的任何与期望违背的结果。

Application Software——应用软件

满足特定需要的软件。

Architecture——构架

一个系统或组件的组织结构。

ASQ——自动化软件质量(Automated Software Quality)

使用软件工具来提高软件的质量。

Assertion——断言

指定一个程序必须已经存在的状态的一个逻辑表达式,或者一组程序变量在程序执行期间的某个点上必须满足的条件。

Assertion Checking——断言检查

用户在程序中嵌入的断言的检查。

Audit——审计

一个或一组工作产品的独立检查以评价与规格、标准、契约或其他准则的符合程度。

Audit Trail——审计跟踪

系统审计活动的一个时间记录。

Automated Testing——自动化测试

使用自动化测试工具来进行测试，这类测试一般不需要人干预，通常在GUI、性能等测试中用得较多。

B.

Backus-Naur Form——BNF范式

一种分析语言，用于形式化描述语言的语法。

Baseline——基线

一个已经被正式评审和批准的规格或产品，它作为进一步开发的一个基础，并且必须通过正式的变更流程来变更。

Basic Block——基本块

一个或多个顺序的可执行语句块，不包含任何分支语句。

Basis Test Set——基本测试集

根据代码逻辑引出来的一个测试用例集合，它保证能获得100%的分支覆盖。

Behaviour——行为

对于一个系统的一个函数的输入和预置条件组合以及需要的反应。一个函数的所有规格包含一个或多个行为。

Benchmark——标杆/指标/基准

一个标准，根据该标准可以进行度量或比较。

Beta Testing——Beta测试

在客户场地，由客户进行的对产品预发布版本的测试。这个测试一般是不可控的。

Big-Bang Testing——大锤测试/一次性集成测试

非渐增式集成测试的一种策略，测试的时候把所有系统的组件一次性组合成系统进行测试。

Black Box Testing——黑盒测试

根据软件的规格对软件进行的测试，这类测试不考虑软件内部的运作原理，因此软件对用户来说就像一个黑盒子。

Bottom-Up Testing——由底向上测试

渐增式集成测试的一种，其策略是先测试底层的组件，然后逐步加入较高层次的组件进行测试，直到系统所有组件都加入系统。

Boundary Value——边界值

一个输入或输出值，它处在等价类的边界上。

Boundary Value Coverage——边界值覆盖

通过测试用例，测试组件等价类的所有边界值。

Boundary Value Testing——边界值测试

通过边界值分析方法来生成测试用例的一种测试策略。

Boundry Value Analysis——边界值分析

该分析一般与等价类一起使用。经验认为软件的错误经常在输入的边界上产生，因此边界值分析就是分析软件输入边界的一种方法。

Branch——分支

在组件中，控制从任何语句到其他任何非直接后续语句的一个条件转换，或者是一个无条件转换。

Branch Condition——分支条件

Branch Condition Combination Coverage——分支条件组合覆盖

在每个判定中所有分支条件结果组合被测试用例覆盖到的百分比。

Branch Condition Combination Testing——分支条件组合测试

通过执行分支条件结果组合来设计测试用例的一种方法。

Branch Condition Coverage——分支条件覆盖

每个判定中分支条件结果被测试用例覆盖到的百分比。

Branch Condition Testing——分支条件测试

通过执行分支条件结果来设计测试用例的一种方法。

Branch Coverage——分支覆盖

通过测试执行到的分支的百分比。

Branch Outcome——分支结果

见判定结果(Decision Outcome)。

Branch Point——分支点

见判定(Decision)。

Branch Testing——分支测试

通过执行分支结果来设计测试用例的一种方法。

Breadth Testing——广度测试

在测试中测试一个产品的所有功能，但是不测试更细节的特性。

Bug——缺陷

C.

Capture/Playback Tool——捕获/回放工具

参考 Capture/Replay Tool。

Capture/Replay Tool——捕获/回放工具

一种测试工具，能够捕获在测试过程中传递给软件的输入，并且能够在以后的时间中，重复这个执行的过程。这类工具一般在 GUI 测试中用得较多。

CASE——计算机辅助软件工程(Computer Aided Software Engineering)

用于支持软件开发的一个自动化系统。

CAST——计算机辅助测试

在测试过程中使用计算机软件工具进行辅助的测试。

Cause-Effect Graph——因果图

一个图形，用来表示输入(原因)与结果之间的关系，可以被用来设计测试用例。

Certification——证明

一个过程，用于确定一个系统或组件与特定的需求相一致。

Change Control——变更控制

一个用于计算机系统或系统数据修改的过程，该过程是质量保证程序的一个关键子集，需要被明确地描述。

Code Audit——代码审计

由一个人、组或工具对源代码进行的一个独立的评审，以验证其与设计规格、程序标准的一致性。正确性和有效性也会被评价。

Code Coverage——代码覆盖率

一种分析方法，用于确定在一个测试套执行后，软件的哪些部分被执行到了，哪些部分没有被执行到。

Code Inspection——代码检视

一个正式的同行评审手段，在该评审中，作者的同行根据检查表对程序的逻辑进行提问，并检查其与编码规范的一致性。

Code Walkthrough——代码走读

一个非正式的同行评审手段，在该评审中，代码被使用一些简单的测试用例进行人工执行，程序变量的状态被手工分析，以分析程序的逻辑和假设。

Code-Based Testing——基于代码的测试

根据从实现中引出的目标设计测试用例。

Coding Standards——编程规范

一些编程方面需要遵循的标准，包括命名方式、排版格式等内容。

Compatibility Testing——兼容性测试

测试软件是否和系统的其他与之交互的元素之间兼容，如浏览器、操作系统、硬件等。

Complete Path Testing——完全路径测试

参考穷尽测试(Exhaustive Testing)。

Completeness——完整性

实体的所有必需部分必须被包含的属性。

Complexity——复杂性

系统或组件难于理解或验证的程度。

Component——组件

一个最小的软件单元，有着独立的规格。

Component Testing——组件测试

参考单元测试。

Computation Data Use——计算数据使用

一个不在条件中的数据使用。

Computer System Security——计算机系统安全性

计算机软件和硬件对偶然的或故意的访问、使用、修改或破坏的一种保护机制。

Condition——条件

一个不包含布尔操作的布尔表达式，例如：A。

Condition Coverage——条件覆盖

通过测试执行到的条件的百分比。

Condition Outcome——条件结果

条件为真为假的评价。

Configuration Control——配置控制

配置管理的一个方面，包括评价、协调、批准和实现配置项的变更。

Configuration Management——配置管理

一套技术和管理方面的原则用于确定和文档化一个配置项的功能和物理属性、控制对这些属性的变更、记录和报告变更处理和实现的状态，以及验证与指定需求的一致性。

Conformance Criterion——一致性标准

判断组件在一个特定输入值上的行为是否符合规格的一种方法。

Conformance Testing——一致性测试

测试一个系统的实现是否和其基于的规格相一致的测试。

Consistency——一致性

在系统或组件的各组成部分和文档之间没有矛盾，一致的程度。

Consistency Checker——一致性检查器

一个软件工具，用于测试设计规格中需求的一致性和完整性。

Control Flow——控制流

程序执行中所有可能的事件顺序的一个抽象表示。

Control Flow Graph——控制流图

通过一个组件的可能替换控制流路径的一个图形表示。

Conversion Testing——转换测试

用于测试已有系统的数据是否能够转换到替代系统上的一种测试。

Corrective Maintenance——故障检修

用于纠正硬件或软件中故障的维护。

Correctness——正确性

软件遵从其规格的程度。

Correctness——正确性

软件在其规格、设计和编码中没有故障的程度。软件、文档和其他项满足需求的程度。软件、文档和其他项满足用户明显的和隐含的需求的程度。

Coverage——覆盖率

用于确定测试所执行到的覆盖项的百分比。

Coverage Item——覆盖项

作为测试基础的一个入口或属性：如语句、分支、条件等。

Crash——崩溃

计算机系统或组件突然并完全地丧失功能。

Criticality——关键性

需求、模块、错误、故障、失效或其他项对一个系统的操作或开发影响的程度。

Criticality Analysis——关键性分析

需求的一种分析,它根据需求的风险情况给每个需求项分配一个关键级别。

Cyclomatic Complexity——循环复杂度

一个程序中独立路径的数量。

D.

Data Corruption——数据污染

违背数据一致性的情况。

Data Definition——数据定义

一个可执行语句,在该语句上一个变量被赋予了一个值。

Data Definition C-Use Coverage——数据定义 C-Use 覆盖

在组件中被测试执行到的数据定义 C-Use 使用对的百分比。

Data Definition C-Use Pair——数据定义 C-Use 使用对

一个数据定义和一个计算数据使用,数据使用的值是数据定义的值。

Data Definition P-Use Coverage——数据定义 P-Use 覆盖

在组件中被测试执行到的数据定义 P-Use 使用对的百分比。

Data Definition P-Use Pair——数据定义 P-Use 使用对

一个数据定义和一个条件数据使用,数据使用的值是数据定义的值。

Data Definition-Use Coverage——数据定义使用覆盖

在组件中被测试执行到的数据定义使用对的百分比。

Data Definition-Use Pair——数据定义使用对

一个数据定义和一个数据使用,数据使用的值是数据定义的值。

Data Definition-Use Testing——数据定义使用测试

以执行数据定义使用对为目标进行测试用例设计的一种技术。

Data Dictionary——数据字典

(1) 一个软件系统中使用的所有数据项名称,以及这些项相关属性的集合。

(2) 数据流、数据元素、文件、数据基础和相关处理的一个集合。

Data Flow Analysis——数据流分析

一个软件验证和确认过程,用于保证输入和输出数据和它们的格式是被适当定义的,并且数据流是正确的。

Data Flow Coverage——数据流覆盖

测试覆盖率的度量是根据变量在代码中的使用情况。

Data Flow Diagram——数据流图

把数据源、数据接受、数据存储和数据处理作为节点描述的一个图形,数据之间的逻辑体现为节点之间的边。

Data Flow Testing——数据流测试

根据代码中变量的使用情况进行的测试。

Data Integrity——数据完整性

一个数据集合完全、正确和一致的程度。

Data Use——数据使用

一个可执行的语句，在该语句中，变量的值被访问。
Data Validation——数据确认
用于确认数据不正确、不完整和不合理的过程。
Dead Code——死代码
在程序操作过程中永远不可能被执行到的代码。
Debugging——调试
发现和去除软件失效根源的过程。
Decision——判定
一个程序控制点，在该控制点上，控制流有两个或多个可替换路由。
Decision Condition——判定条件
判定内的一个条件。
Decision Coverage——判定覆盖
在组件中被测试执行到的判定结果的百分比。
Decision Outcome——判定结果
一个判定的结果，决定控制流走哪条路径。
Decision Table——判定表
一个表格，用于显示条件和条件导致动作的集合。
Depth Testing——深度测试
执行一个产品的一个特性的所有细节，但不测试所有特性。比较广度测试。
Design Of Experiments——实验设计
一种计划实验的方法，这样适合分析的数据可以被收集。
Design-Based Testing——基于设计的测试
根据软件的构架或详细设计引出测试用例的一种方法。
Desk Checking——桌面检查
通过手工模拟软件执行的方式进行测试的一种方式。
Diagnostic——诊断
检测和隔离故障或失效的过程。
Dirty Testing——肮脏测试
参考负面测试(Negative Testing)。
Disaster Recovery——灾难恢复
一个灾难的恢复和重建过程或能力。
Documentation Testing——文档测试
测试关注于文档的正确性。
Domain——域
值被选择的一个集合。
Domain Testing——域测试
参考等价划分测试(Equivalence Partition Testing)。
Dynamic Analysis——动态分析
根据执行的行为评价一个系统或组件的过程。

Dynamic Testing——动态测试

通过执行软件的手段来测试软件。

E.

Embedded Software——嵌入式软件

软件运行在特定硬件设备中,不能独立于硬件存在。这类系统一般要求实时性较高。

Emulator——仿真

一个模仿另一个系统的系统或设备,它接收相同的输入并产生相同的输出。

End-To-End Testing——端到端测试

在一个模拟现实使用的场景下测试一个完整的应用环境,例如和数据库交互,使用网络通信等。

Entity Relationship Diagram——实体关系图

描述现实世界中实体及它们关系的图形。

Entry Point——入口点

一个组件的第一个可执行语句。

Equivalence Class——等价类

组件输入或输出域的一个部分,在该部分中,组件的行为从组件的规格上来看认为是相同的。

Equivalence Partition Coverage——等价划分覆盖

在组件中被测试执行到的等价类的百分比。

Equivalence Partition Testing——等价划分测试

根据等价类设计测试用例的一种技术。

Equivalence Partitioning——等价划分

组件的一个测试用例设计技术,该技术从组件的等价类中选取典型的点进行测试。

Error——错误

IEEE 的定义是:一个人为产生不正确结果的行为。

Error Guessing——错误猜测

根据测试人员以往的经验猜测可能出现问题的地方来进行用例设计的一种技术。

Error Seeding——错误播种/错误插值

故意插入一些已知故障(Fault)到一个系统中去的过程,目的是为了根据错误检测和跟踪的效率并估计系统中遗留缺陷的数量。

Exception——异常/例外

一个引起正常程序执行挂起的事件。

Executable Statement——可执行语句

一个语句在被编译后会转换成目标代码,当程序运行时会被执行,并且可能对程序数据产生动作。

Exhaustive Testing——穷尽测试

测试覆盖软件的所有输入和条件组合。

Exit Point——出口点

一个组件的最后一个可执行语句。

Expected Outcome——期望结果

参考预期结果(Predicted Outcome)。

F.

Failure——失效

软件的行为与其期望的服务相背离。

Fault——故障

在软件中一个错误的表现。

Feasible Path——可达路径

可以通过一组输入值和条件执行到的一条路径。

Feature Testing——特性测试

参考功能测试(Functional Testing)。

FMEA——失效模型效果分析(Failure Modes And Effects Analysis)

可靠性分析中的一种方法,用于在基本组件级别上确认对系统性能有重大影响的失效。

FMECA——失效模型效果关键性分析(Failure Modes And Effects Criticality Analysis)

FMEA 的一个扩展,它分析了失效结果的严重性。

FTA——故障树分析(Fault Tree Analysis)

引起一个不需要事件产生的条件和因素的确认和分析,通常是严重影响系统性能、经济性、安全性或其他需要特性。

Functional Decomposition——功能分解

参考模块分解(Modular Decomposition)

Functional Specification——功能规格说明书

一个详细描述产品特性的文档。

Functional Testing——功能测试

测试一个产品的特性和可操作行为以确定它们满足规格。

G.

Glass Box Testing——玻璃盒测试

参考白盒测试(White Box Testing)。

I.

IEEE——美国电子与电器工程师学会(Institute Of Electrical And Electronic Engineers)

Incremental Testing——渐增测试

集成测试的一种,组件逐渐被增加到系统中直到整个系统被集成。

Infeasible Path——不可达路径

不能够通过任何可能的输入值集合执行到的路径。

Input Domain——输入域

所有可能输入的集合。

Inspection——检视

对文档进行的一种评审形式。

Installability Testing——可安装性测试

确定系统的安装程序是否正确的测试。

Instrumentation——插装

在程序中插入额外的代码以获得程序在执行时行为的信息。

Instrumenter——插装器

执行插装的工具。

Integration Testing——集成测试

测试一个应用组合后的部分以确保它们的功能在组合之后正确。该测试一般在单元测试之后进行。

Interface——接口

两个功能单元的共享边界。

Interface Analysis——接口分析

分析软件与硬件、用户和其他软件之间接口的需求规格。

Interface Testing——接口测试

测试系统组件间接口的一种测试。

Invalid Inputs——无效输入

在程序功能输入域之外的测试数据。

Isolation Testing——孤立测试

组件测试(单元测试)策略中的一种,把被测组件从其上下文组件之中孤立出来,通过设计驱动和桩进行测试的一种方法。

J.

Job——工作

一个用户定义的要计算机完成的工作单元。

Job Control Language——工作控制语言

用于确定工作顺序,描述它们对操作系统要求并控制它们执行的语言。

L.

LCSAJ——线性代码顺序和跳转(Linear Code Sequence And Jump)

包含三个部分:可执行语句线性顺序的起始,线性顺序的结束,在线性顺序结束处控制流跳转的目标语句。

LCSAJ Coverage——LCSAJ 覆盖

在组件中被测试执行到的 LCSAJ 的百分比。

LCSAJ Testing——LCSAJ 测试

根据 LCSAJ 设计测试用例的一种技术。

Load Testing——负载测试

通过测试系统在资源超负荷情况下的表现,以发现设计上的错误或验证系统的负载能力。

Logic Analysis——逻辑分析

(1) 评价软件设计的关键安全方程式、算法和控制逻辑的方法。

(2) 评价程序操作的顺序并且检测可能导致灾难的错误。

Logic-Coverage Testing——逻辑覆盖测试

参考结构化测试用例设计(Structural Test Case Design)。

M.

Maintainability——可维护性

一个软件系统或组件可以被修改的容易程度，这个修改一般是因为缺陷纠正、性能改进或特性增加引起的。

Maintainability Testing——可维护性测试

测试系统是否满足可维护性目标。

Modified Condition/Decision Coverage——修改条件/判定覆盖

在组件中被测试执行到的修改条件/判定的百分比。

Modified Condition/Decision Testing——修改条件/判定测试

根据 MC/DC 设计测试用例的一种技术。

Monkey Testing——跳跃式测试

随机性，跳跃式地测试一个系统，以确定一个系统是否会崩溃。

MTBF——平均失效间隔时间(Mean Time Between Failures)

两次失效之间的平均操作时间。

MTTF——平均失效时间(Mean Time To Failure)

第一次失效之前的平均时间。

MTTR——平均修复时间(Mean Time To Repair)

两次修复之间的平均时间。

Multiple Condition Coverage——多条件覆盖

参考分支条件组合覆盖(Branch Condition Combination Coverage)。

Mutation Analysis——变体分析

一种确定测试用例套完整性的方法，该方法通过判断测试用例套能够区别程序与其变体之间的程度。

N.

Negative Testing——逆向测试/反向测试/负面测试

测试瞄准于使系统不能工作。

Non-Functional Requirements Testing——非功能性需求测试

与功能不相关的需求测试，如性能测试、可用性测试等。

N-Switch Coverage——N 切换覆盖

在组件中被测试执行到的 N 转换顺序的百分比。

N-Switch Testing——N 切换测试

根据 N 转换顺序设计测试用例的一种技术，经常用于状态转换测试中。

N-Transitions——N 转换

N+1 转换顺序。

O.

Operational Testing——可操作性测试

在系统或组件操作的环境中评价它们的表现。

Output Domain——输出域

所有可能输出的集合。

P.

Partition Testing——分类测试

参考等价划分测试(Equivalence Partition Testing)。

Path——路径

一个组件从入口到出口的一条可执行语句顺序。

Path Coverage——路径覆盖

在组件中被测试执行到的路径的百分比。

Path Sensitizing——路径敏感性

选择一组输入值强制组件走一个给定的路径。

Path Testing——路径测试

根据路径设计测试用例的一种技术,经常用于状态转换测试中。

Performance Testing——性能测试

评价一个产品或组件与性能需求是否符合的测试。

Portability Testing——可移植性

测试瞄准于证明软件可以被移植到指定的硬件或软件平台上。

Positive Testing——正向测试

测试瞄准于显示系统能够正常工作。

Precondition——预置条件

环境或状态条件,组件执行之前必须被填充一个特定的输入值。

Predicate——谓词

一个逻辑表达式,结果为'真'或'假'。

Predicate Data Use——谓词数据使用

在谓词中的一个数据使用。

Program Instrumenter——程序插装

参考插装(Instrumenter)。

Progressive Testing——递进测试

在先前特性回归测试之后对新特性进行测试的一种策略。

Pseudo-Random——伪随机

看似随机的,实际上是根据预先安排的顺序进行的。

Q.

QA——质量保证(Quality Assurance)

(1) 已计划的系统性活动,用于保证一个组件、模块或系统遵从已确立的需求。

(2) 采取的所有活动以保证一个开发组织交付的产品满足性能需求和已确立的标准和过程。

QC——质量控制(Quality Control)

用于获得质量需求的操作技术和过程,如测试活动。

R.

Race Condition——竞争状态

并行问题的根源。对一个共享资源的多个访问,至少包含一个写操作,但是没有一个机

制来协调同时发生的访问。

Recovery Testing——恢复性测试

验证系统从失效中恢复能力的测试。

Regression Analysis and Testing——回归分析和测试

一个软件验证和确认任务以确定在修改后需要重复测试和分析的范围。

Regression Testing——回归测试

在发生修改之后重新测试先前的测试以保证修改的正确性。

Release——发布

一个批准版本的正式通知和分发。

Reliability——可靠性

一个系统或组件在规定的条件下在指定的时间内执行其需要功能的能力。

Reliability Assessment——可靠性评价

确定一个已有系统或组件的可靠性级别的过程。

Requirements-Based Testing——基于需求的测试

根据软件组件的需求导出测试用例的一种设计方法。

Review——评审

在产品开发过程中，把产品提交给项目成员、用户、管理者或其他相关人员评价或批准的过程。

Risk——风险

不期望效果的可能性和严重性的一个度量。

Risk Assessment——风险评估

对风险和风险影响的一个完整的评价。

S.

Safety——(生命)安全性

不会引起人员伤亡、产生疾病、毁坏或损失设备和财产或者破坏环境。

Safety Critical——严格的安全性

一个条件、事件、操作、过程或项，它的认识、控制或执行对生命安全性的系统来说是非常关键的。

Sanity Testing——理智测试

软件主要功能成分的简单测试以保证它是否能进行基本的测试。参考冒烟测试。

SDP——软件开发计划(Software Development Plan)

用于一个软件产品开发的项目计划。

Security Testing——安全性测试

验证系统是否符合安全性目标的一种测试。

Security——(信息)安全性

参考计算机系统安全性(Computer System Security)

Serviceability Testing——可服务性测试

参考可维护性测试(Maintainability Testing)

Simple Subpath——简单子路径

控制流的一个子路径，其中没有不必要的部分被执行。

Simulation——模拟

使用另一个系统来表示一个物理的或抽象的系统的选定行为特性。

Simulation——模拟

使用一个可执行模型来表示一个对象的行为。

Simulator——模拟器

软件验证期间的一个设备、软件程序或系统，当它给定一个控制的输入时，表现的与一个给定的系统类似。

SLA——服务级别协议(Service Level Agreement)

服务提供商与客户之间的一个协议，用于规定服务提供商应当提供什么服务。

Smoke Testing——冒烟测试

对软件主要功能进行快餐式测试。最早来自于硬件测试实践，以确定新的硬件在第一次使用的时候不会着火。

Software Development Process——软件开发过程

一个把用户需求转换为软件产品的开发过程。

Software Diversity——软件多样性

一种软件开发技术，其中是由不同的程序员或开发组开发的相同规格的不同程序，目的是为了检测错误、增加可靠性。

Software Element——软件元素

软件开发或维护期间产生或获得的一个可交付的或过程内的文档。

Software Engineering——软件工程

一个应用于软件开发、操作和维护的系统性的、有纪律的、可量化的方法。

Software Engineering Environment——软件工程环境

执行一个软件工程工作的硬件、软件和固件。

Software Life Cycle——软件生命周期

开始于一个软件产品的构思，结束于该产品不再被使用的这段时间。

SOP——标准操作过程(Standard Operating Procedures)

书面的步骤，这对保证生产和处理的控制是必需的。

Source Code——源代码

一种适合输入到汇编器、编译器或其他转换设备的计算机指令和数据定义。

Source Statement——源语句

参考语句(Statement)。

Specification——规格

组件功能的一个描述，格式是：对指定的输入在指定的条件下的输出。

Specified Input——指定的输入

一个输入，根据规格能预知其输出。

Spiral Model——螺旋模型

软件开发过程的一个模型，其中的组成活动，典型的包括需求分析，概要设计，详细设计，编码，集成和测试等活动被迭代地执行直到软件被完成。

SQL——结构化查询语句(Structured Query Language)

在一个关系数据库中查询和处理数据的一种语言。

State——状态

一个系统、组件或模拟可能存在其中的一个条件或模式。

State Diagram——状态图

一个图形,描绘一个系统或组件可能假设的状态,并且显示引起或导致一个状态切换到另一个状态的事件或环境。

State Transition——状态转换

一个系统或组件的两个允许状态之间的切换。

State Transition Testing——状态转换测试

根据状态转换来设计测试用例的一种方法。

Statement——语句

程序语言的一个实体,是典型的最小可执行单元。

Statement Coverage——语句覆盖

在一个组件中,通过执行一定的测试用例所能达到的语句覆盖百分比。

Statement Testing——语句测试

根据语句覆盖来设计测试用例的一种方法。

Static Analysis——静态分析

分析一个程序的执行,但是并不实际执行这个程序。

Static Analyzer——静态分析器

进行静态分析的工具。

Static Testing——静态测试

不通过执行来测试一个系统。

Statistical Testing——统计测试

通过使用对输入统计分布进行分析来构造测试用例的一种测试设计方法。

Stepwise Refinement——逐步优化

一个结构化软件设计技术,数据和处理步骤首先被广泛地定义,然后被逐步地进行了细化。

Storage Testing——存储测试

验证系统是否满足指定存储目标的测试。

Stress Testing——压力测试

在规定的规格条件或者超过规定的规格条件下,测试一个系统,以评价其行为。类似负载测试,通常是性能测试的一部分。

Structural Coverage——结构化覆盖

根据组件内部的结构度量覆盖率。

Structural Test Case Design——结构化测试用例设计

根据组件内部结构的分析来设计测试用例的一种方法。

Structural Testing——结构化测试

参考结构化测试用例设计(Structural Test Case Design)

Structured Basis Testing——结构化的基础测试

根据代码逻辑设计测试用例来获得100%分支覆盖的一种测试用例设计技术。

Structured Design——结构化设计

软件设计的任何遵循一定纪律的方法，它按照特定的规则，例如：模块化，由顶向下设计，数据逐步优化，系统结构和处理步骤。

Structured Programming——结构化编程

在结构化程序开发中的任何包含结构化设计和结果的软件开发技术。

Structured Walkthrough——结构化走读

参考走读(Walkthrough)。

Stub——桩

一个软件模块的框架或特殊目标实现，主要用于开发和测试一个组件，该组件调用或依赖这个模块。

Symbolic Evaluation——符号评价

参考符号执行(Symbolic Execution)。

Symbolic Execution——符号执行

通过符号表达式来执行程序路径的一种静态分析设计技术。其中，程序的执行被用符号来模拟，例如，使用变量名而不是实际值，程序的输出被表示成包含这些符号的逻辑或数学表达式。

Symbolic Trace——符号轨迹

一个计算机程序通过符号执行时经过的语句分支结果的一个记录。

Syntax Testing——语法分析

根据输入语法来验证一个系统或组件的测试用例设计技术。

System Analysis——系统分析

对一个计划的或现实的系统进行的一个系统性调查以确定系统的功能以及系统与其他系统之间的交互。

System Design——系统设计

一个定义硬件和软件构架、组件、模块、接口和数据的过程以满足指定的规格。

System Integration——系统集成

一个系统组件的渐增的连接和测试，直到一个完整的系统。

System Testing——系统测试

从一个系统的整体而不是个体上来测试一个系统，并且该测试关注的是规格，而不是系统内部的逻辑。

T.

Technical Requirements Testing——技术需求测试

参考非功能需求测试(Non-Functional Requirements Testing)

Test Automation——测试自动化

使用工具来控制测试的执行、结果的比较、测试预置条件的设置、其他测试控制和报告功能。

Test Case——测试用例

用于特定目标而开发的一组输入、预置条件和预期结果。

Test Case Design Technique——测试用例设计技术

选择和导出测试用例的技术。

Test Case Suite——测试用例套

对被测软件的一个或多个测试用例的集合。

Test Comparator——测试比较器

一个测试工具用于比较软件实际测试产生的结果与测试用例预期的结果。

Test Completion Criterion——测试完成标准

一个标准用于确定被计划的测试何时完成。

Test Coverage——测试覆盖

参考覆盖率(Coverage)

Test Driver——测试驱动

一个程序或测试工具用于根据测试套执行软件。

Test Environment——测试环境

测试运行其上的软件和硬件环境的描述，以及任何其他与被测软件交互的软件，包括驱动和桩。

Test Execution——测试执行

一个测试用例被被测软件执行，并得到一个结果。

Test Execution Technique——测试执行技术

执行测试用例的技术，包括手工、自动化等。

Test Generator——测试生成器

根据特定的测试用例产生测试用例的工具。

Test Harness——测试用具

包含测试驱动和测试比较器的测试工具。

Test Log——测试日志

一个关于测试执行所有相关细节的时间记录。

Test Measurement Technique——测试度量技术

度量测试覆盖率的技术。

Test Plan——测试计划

一个文档，描述了要进行的测试活动的范围、方法、资源和进度。它确定测试项、被测特性、测试任务、谁执行任务，并且任何风险都要冲突计划。

Test Procedure——测试规程

一个文档，提供详细的测试用例执行指令。

Test Records——测试记录

对每个测试，明确地记录被测组件的标识、版本，测试规格和实际结果。

Test Report——测试报告

一个描述系统或组件执行的测试和结果的文档。

Test Script——测试脚本

一般指的是一个特定测试的一系列指令，这些指令可以被自动化测试工具执行。

Test Specification——测试规格

一个文档,用于指定一个软件特性、特性组合或所有特性的测试方法、输入、预期结果和执行条件。

Test Strategy——测试策略

一个简单的高层文档,用于描述测试的大致方法、目标和方向。

Test Suite——测试套

测试用例和/或测试脚本的一个集合,与一个应用的特定功能或特性相关。

Test Target——测试目标

一组测试完成标准。

Testability——可测试性

一个系统或组件有利于测试标准建立和确定这些标准是否被满足的测试执行的程度。

Testing——测试

IEEE 给出的定义是:①一个执行软件的过程,以验证其满足指定的需求并检测错误。②一个软件项的分析过程以检测已有条件之间的不同,并评价软件项的特性。

Thread Testing——线程测试

自顶向下测试的一个变化版本,其中,递增的组件集成遵循需求子集的实现。

Time Sharing——时间共享

一种操作方式,允许两个或多个用户在相同的计算机系统上同时执行计算机程序。其实现可能通过时间片轮转、优先级中断等。

Top-Down Design——由顶向下设计

一种设计策略,首先设计最高层的抽象和处理,然后逐步向更低级别进行设计。

Top-Down Testing——自顶向下测试

集成测试的一种策略,首先测试最顶层的组件,其他组件使用桩,然后逐步加入较低层的组件进行测试,直到所有组件被集成到系统中。

Traceability——可跟踪性

开发过程的两个或多个产品之间关系可以被建立起来的程度,尤其是产品彼此之间有一个前后处理关系。

Traceability Analysis——跟踪性分析

(1) 跟踪概念文档中的软件需求到系统需求;

(2) 跟踪软件设计描述到软件需求规格,以及软件需求规格到软件设计描述;

(3) 跟踪源代码对应到设计规格,以及设计规格对应到源代码。分析确定它们之间正确性、一致性、完整性、精确性的关系。

Traceability Matrix——跟踪矩阵

一个用于记录两个或多个产品之间关系的矩阵。例如,需求跟踪矩阵是跟踪从需求到设计再到编码的实现。

Transaction——事务/处理

(1) 一个命令、消息或输入记录,它明确或隐含地调用了一个处理活动,例如更新一个文件。

(2) 用户和系统之间的一次交互。

(3) 在一个数据库管理系统中，完成一个特定目的的处理单元，如恢复、更新、修改或删除一个或多个数据元素。

Transform Analysis——事务分析

系统的结构是根据分析系统需要处理的事务获得的一种分析技术。

Trojan Horse——特洛伊木马

一种攻击计算机系统的方法，典型的方法是提供一个包含具有攻击性隐含代码的有用程序给用户，在用户执行该程序的时候，其隐含的代码对系统进行非法访问，并可能产生破坏。

Truth Table——真值表

用于逻辑操作的一个操作表格。

U.

Unit Testing——单元测试

测试单个的软件组件，属于白盒测试范畴，其测试基础是软件内部的逻辑。

Usability Testing——可用性测试

测试用户使用和学习产品的容易程度。

V.

Validation——确认

根据用户需要确认软件开发的产品的正确性。

Verification——验证

评价一个组件或系统以确认给定开发阶段的产品是否满足该阶段开始时设定的标准。

Version——版本

一个软件项或软件元素的一个初始发布或一个完整的再发布。

Volume Testing——容量测试

使用大容量数据测试系统的一种策略。

W.

Walkthrough——走读

一个针对需求、设计或代码的非正式的同行评审，一般由作者发起，由作者的同行参与进行的评审过程。

Waterfall Model——瀑布模型

软件开发过程模型的一种，包括概念阶段、需求阶段、设计阶段、实现阶段、测试阶段、安装和检查阶段、操作和维护阶段，这些阶段按次序进行，可能有部分重叠，但很少会迭代。

White Box Testing——白盒测试

根据软件内部的工作原理分析来进行测试。



软件测试常见问题

软件测试的目标是要发现错误,因此在编写测试用例的时候也要遵循这个目标,尽量在软件的最薄弱环节多编写测试用例。虽然测试时有很多单个输入变量、多个输入变量的组合,但优秀的软件测试人员不会依靠运气,他们有着丰富的经验和直觉,可以从中找到哪些是需要进行测试的,哪些是不需要测试的,哪些操作可能会引起软件失效。把这些测试人员的经验和直觉尽量归纳和固化,就形成了下面介绍的21种故障模型。故障模型指明了故障是如何以及为什么会在软件执行时引起软件失效。在测试过程中,可以按照这些故障模型所提供的缺陷类型和寻找该类缺陷的方法找到尽量多的缺陷。

1. 输入非法数据

1) 缺陷产生原因

开发人员通常用以下三种技术来处理非法输入。

(1) 防止不正确的输入进入被测软件。过滤掉不正确的输入,只允许合法输入通过界面。

(2) 输入不正确的数据后,软件提示错误信息,拒绝不正确的输入。

(3) 允许不正确的输入进入系统并进行处理,软件失效时调用异常处理程序,显示一些错误信息。

可见开发人员除了编写主要的功能代码外,还必须编写对非法输入的检查代码,这些代码经常被遗忘,或者编写完这部分代码后,开发人员很少认真检查,导致处理非法输入经常出错。

2) 如何发现这类问题

进行测试时从输入值的属性出发,一般考虑以下三点。

(1) 输入类型:输入无效的类型常会产生错误信息。

(2) 输入长度:对于字符型,输入太多的字符常会引出错误信息。

(3) 边界值:输入边界值或超过边界值的数据。

3) 测试方法小结

(1) 应用场合:GUI的输入。

(2) 测试方法:分别从输入数据的类型、长度、边界值等方面进行考虑。

(3) 测试信息检查:

① 错误信息和错误要一致。

② 错误信息的内容为空,用户不知道为什么出错。

③ 显示的错误信息是给开发人员调试使用的，例如“Error 5-unknown data”，开发人员可以通过该信息很容易找到错误类型，但是用户根本不明白，不知道做错了什么。

（4）测试知识储备：牢记各基本数据类型的边界值。

2. 输入默认值

1）缺陷产生原因

一旦软件中使用了变量，就必须赋给初始值，如果在赋值之前就使用了这些变量，软件就会失效，正确地使用变量的顺序是：声明变量→给变量赋值→使用变量。通常会由于以下两个原因使变量的默认值不正确。

（1）给变量赋值这一步经常会被开发人员不经意地略过。

（2）开发人员有时不确定到底要赋什么初始值，就随便给了一个值，但用户并不认可该值，这种情况下，软件并不一定会失效，但对用户的使用会带来很多不便。例如，某程序把打印默认输出份数设置为两份，会给用户造成很大麻烦。

2）如何发现这类问题

确定应用软件中所使用的数据有以下一些基本原则。

（1）查找选项按钮、配置面板、安装屏幕等。这种屏幕上显示的数据常常在应用程序的许多地方用到。

（2）查阅源代码的数据声明部分（如果可以得到）。

（3）确定了要测试的数据，可以通过以下操作来强制使用或不使用默认的值。

① 接受软件显示的默认值。有时软件需要用户输入一个值，如果没有输入任何值，软件就可能失效。这时可以只是简单地单击“确定”按钮来接受默认值，完成这个功能测试。

② 输入空值。删除默认值，使输入域变成空值。

③ 将默认值改为另一个值，这样会使应用程序以不同的值来运行。

④ 将输入值改为另一个值，然后再变以空值。

一个好的软件会这样处理以上情况，将输入的不合法内容默认为合法边界内的某个合理值，或者返回错误提示信息。

3）测试方法小结

（1）应用场合：需要有默认值的地方。

（2）测试方法：分别从选项按钮、配置面板、安装配置、开始界面等方面进行考虑，强制使用或不使用默认值等。

（3）测试知识储备：全面理解需求规格说明书中对默认值的要求；同时深刻理解被测软件的行业背景。

3. 输入特殊字符集

1）缺陷产生原因

应用程序接受字符串输入，如果程序没有针对特殊输入进行特殊编程，那么就有可能导致程序挂起，主要包括以下三种情况。

（1）字符集包括普通字符和特殊字符。例如，ASCII 字符集包括普通字符和特殊字符。应用程序有时只能处理普通字符，当输入特殊字符时就会出现错误。

(2) 实现应用程序的程序设计语言有特定的处理一些字符和字符串的方法。例如,C语言把\n、++和 & 这样的字符用于特殊目的。如果将这些字符串输入到对话框中,程序必须进行错误处理,否则容易产生错误。

(3) 应用程序有时也使用设置名称、系统对象和程序的保留字符串集合。只要在程序中使用了这些字符串,就可能导致失效。

2) 如何发现这类问题

(1) 根据被测软件所处的操作系统、使用的程序设计语言、字符集等信息列出表格,通过测试小组的讨论,标明应用表格中的哪些字符和数据类型作为输入来测试。

(2) 根据经验,软件很少会因为这种操作而崩溃,通常它会挂起没有响应。

3) 测试方法小结

(1) 应用场合:需要接收字符输入的地方。

(2) 测试方法:根据被测软件的具体情况输入非法字符。

(3) 测试知识储备:尽可能多地了解字符集、程序设计语言和操作系统中的保留字符串及其特定含义,可以更好地分辨这类缺陷。

4. 输入使缓冲区溢出的数据

1) 缺陷产生原因

开发人员没有考虑传送给内存缓冲区的字符串的大小。如果缓冲区只能保留固定长度的字符串,输入更长的字符串就会改写其他的内存存储单元,引起操作系统强制性地终止应用程序。

2) 如何发现这类问题

当应用程序允许输入字母、数字时,通过 GUI 控件(如文本框),或者通过 API 调用的参数来进行这种测试。

(1) 弄清楚要测试的输入域的长度,输入最大字符串测试。

(2) 输入一个比最大字符串长的字符串,应用程序可能出现错误提示信息,提示不允许输入;或者输入了更长的字符串使应用程序崩溃。

3) 测试方法小结

(1) 应用场合:需要接收字符输入的地方。

(2) 测试方法:根据被测软件的具体情况输入最大字符串或输入一个比最大字符串更长的字符串。

(3) 测试知识储备:尽可能多地和开发人员讨论,以了解和确定输入域的合理长度。

5. 输入产生错误的合法数据组合

1) 缺陷产生原因

测试多个输入值的组合,每个输入值已被单独测试过,但是这些值的组合可能会互相影响而引起软件失效。

2) 如何发现这类问题

首先要确定测试哪些输入组合,并弄清楚它们之间的"关系"。如果具备以下任一特性,就可以认为这些变量是有"关系"的。

（1）描述的是有关单个内部数据结构的属性和内容。例如，输入面板需要用户输入列表的“行”和“列”，这时测试人员要输入单个内部数据结构“列表”的属性“行”和“列”。

（2）一起用在了一个计算中，也就是将多个输入用作一个内部计算的操作数，因此这些输入变量具有了相互“关系”。

3）测试方法小结

（1）应用场合：输入值之间存在依赖关系。

（2）测试方法：输入可能存在问题的组合值。

（3）测试知识储备：尽可能多地了解内部数据结构的属性和内容，并与开发人员探讨，以确定输入的数据值。

6．产生同一个输入的各种可能输出

1）缺陷产生原因

单个输入产生多种输出的情况与先前的输入和被测系统的状态都有关系。例如，在文字处理程序中单击“关闭”按钮，如果文件被编辑且未被保存，程序将提示是否保存文件。如果文件已被保存过，则文件直接关闭。

2）如何发现这类问题

测试人员必须具有关于被测系统软件的业务方面的知识，具备各种程序文档，明确一个输入可以产生何种输出。可以据此列出关于程序输入与输出的一个列表，然后进行测试。

3）测试方法小结

（1）应用场合：同一输入对应多个输出的情况。

（2）测试方法：测试输入对应的每一个输出。

（3）测试知识储备：全面理解需求规格说明书中的内容，找出输入与输出之间的关系。

7．输出不符合业务规则的无效输出

1）缺陷产生原因

有时开发人员也可能对业务了解不深刻，对有些问题也是一知半解，因此编写出的软件就会产生不符合业务逻辑的问题。另外，在绝大多数情况下开发人员会忽略处理没有遵循一般规则的输入，如果不对这些特殊情况进行编程处理，软件就会产生错误的结果。

2）如何发现这类问题

（1）测试人员应该尽可能地学习涉及问题的领域。

（2）有时在列举出无效输出后，也很难知道哪些输入组合能强制这些输出产生。这时测试人员必须先要确定哪些输入与输出有关，然后用产生意外结果的输入组合进行测试，测试过程中要注意输入执行顺序，用不同的顺序执行可能得到不同的结果。如果不能强制无效的输出产生，就说明软件没有这方面的缺陷。

3）测试方法小结

（1）应用场合：强制产生不符合业务背景的知识。

（2）测试方法：列举出所有的无效输出，然后逐一测试。

（3）测试知识储备：全面理解需求规格说明书中的内容，熟悉行业背景知识。

8. 输出属性修改后的结果

1）缺陷产生原因

输出常常具有可修改的属性，如颜色、形状、维数及大小等，用户可以修改这些属性。在这种情况下，开发人员必须编码、设立初始或默认属性值，然后编码允许用户编辑这些属性。当用户改变了这些属性后，内部的相应变量值也随着变化，再次进行处理时，这些值没有被重新恢复为默认值，输出的属性就被强制改变了。

2）如何发现这类问题

该测试方法可以使用在那些输出具有可编辑性、可修改性的功能中。测试人员首先要仔细了解能够产生的输出，特别要注意具有可编辑属性的输出。测试人员的任务就是强制每个输出产生，并编辑其属性，然后再次强制输出产生。

3）测试方法小结

（1）应用场合：输出的结果，可以由用户修改属性得出。

（2）测试方法：强制每个输出产生，并编辑其属性，然后再次强制产生输出。

（3）测试知识储备：全面理解需求规格说明书中的内容，了解能够产生的输出。

9. 屏幕刷新显示

1）缺陷产生原因

通常 GUI 软件会产生刷新问题，因为 GUI 在对窗口进行覆盖、移动和调整大小时，必须刷新屏幕才能使对象重新显示。但是如果经常刷新，容易减慢应用程序的运行速度；如果不刷新，又会影响用户对程序的使用，使用户必须停止工作，去寻找刷新的方法才可以继续工作。所以开发人员有时候不能很好地确定什么时候需要刷新，需要刷新多大范围的区域，这就发生了令人烦恼的刷新问题。

2）如何发现这类问题

测试刷新问题的方法是增加、删除移动屏幕上的对象，这样会使某些对象重新显示。如果不能正确、及时地进行重新显示，就产生了软件缺陷。可以通过以下几个方法来检查刷新。

（1）从起始位置移动对象。先移动一点儿，然后增加移动幅度；先移动一次或两次，然后多次移动，确保覆盖了所有区域。

（2）从覆盖对象的边界开始一点点儿覆盖，使其中一个对象遮住另一个对象。

（3）使用不同类型的对象。如果应用程序支持多种类型的对象，如文本对象、图形对象等，就把这些不同对象混在一起使用。

3）测试方法小结

（1）应用场合：一个对象包含在另一个对象中，拖动被包含对象时，可能出现刷新问题。

（2）测试方法：增加、删除和移动屏幕上的对象。

（3）测试知识储备：全面理解需求规格说明书中的内容，了解程序中对象之间的关系。

10. 数据结构溢出

1）缺陷产生原因

所有数据结构的大小都有上限。一些数据结构会逐步增加长度以充满机器内存容量或磁盘空间，而其他数据结构具有固定的上限。开发人员经常对有关数据结构的内容进行编码，忘记结构本身的物理局限。

2）如何发现这类问题

（1）确定数据结构的界限，尝试将过多的值输入数据结构。应该特别注意界限为数据类型的边界 256、1024、32 768 等上溢的测试。

（2）对于下溢的测试，可以尝试多删除一个数据，例如当结构为空时，尝试再删除，或者添加一个数据，尝试删除两个数据时的情况。

3）测试方法小结

（1）应用场合：程序中存在数组。

（2）测试方法：尝试将过多的值输入数据结构，测试上溢；对于下溢的测试，可以尝试多删除一个数据。

（3）测试知识准备：全面理解需求规格说明书中的内容，确定数据结构的界限。

11. 数据结构不符合约束

1）缺陷产生原因

在编程过程中对内部数据结构都有所约束，包括大小、维数、类型、形状、屏幕上的位置等。测试的重点就是用户能够设置的属性，这些属性使用了一组参数来约束。在建立数据项和随后对数据项进行修改的任何时刻都要对数据属性的约束进行检查。初始化代码中修改后的代码有错误，在修改错误的时候只修改了初始化部分，而忽略了对其他部分的修改，使得其修改不完全，不彻底。

2）如何发现这类问题

（1）确认候选数据，并列出其可修改的属性。对每个属性列出有效值的允许范围、约束的条件等。

（2）确定所有可修改属性的功能位置。

（3）对数据进行初始化，改变每个属性以确定是否正确进行了约束。

如果数据约束遭到破坏，可能导致系统崩溃，或者表现为响应时间延迟，错误信息不正确以及使用错误数据产生的无效输出。

3）测试方法小结

（1）应用场合：应用程序内部的数据结构存在约束。

（2）测试方法：破坏内部数据结构的约束。

（3）测试知识储备：全面理解需求规格说明书中的内容，确定内部数据结构的所有约束。

12. 操作数与操作符不符

1）缺陷产生原因

几乎每个运算符都有它无效的操作数，对于具体的操作符，开发人员在使用它们时，必

须编写错误检查代码。例如：除以零的问题。

2）如何发现这类问题

找到程序中包含的数据或输入（即操作数）的计算（即操作符）、数学表达式（即操作符和操作数的组合）及对图形的操作。另外，对多个操作数进行组合也更容易发生错误。例如，字符和数字都可以使用“+”操作符。对字符通过“+”把它们连成一串；对数字通过“+”来进行加法运算。如果系统尝试把字符和数字相加，即进行相互矛盾的操作，就会引起软件失效。

3）测试方法小结

（1）应用场合：需要进行数值计算的程序或图形操作的程序。

（2）测试方法：对于数值计算考虑操作数和操作符之间的限定关系，对于图形计算还要考虑各种输入数据之间的组合关系。

（3）测试知识储备：全面掌握被测软件中操作符对操作数的要求。掌握不同的操作符和操作数具有的不同的有效和无效的取值范围。

13. 递归调用自身

1）缺陷产生原因

函数有时会递归调用自身，如果不限制执行次数，递归就会出现问题，它不断地调用自身，很快地占用机器资源，最终产生溢出，使程序崩溃或挂起。产生这类问题的主要原因是开发人员没有编码来保证循环和递归调用的终止，通常是在循环的开始或结束时缺少检查条件。

2）如何发现问题

在软件中寻找可以使用递归调用的功能。这时可以制作一个列表，标明软件中可能嵌入递归的功能的列表，然后自己引用自己来检查程序是否能正确处理。

3）测试方法小结

（1）应用场合：需要和其他对象进行交互的地方。

（2）测试方法：考虑对象的自我交互或复制。

（3）测试知识储备：全面掌握被测软件的需求。

14. 计算结果溢出

1）缺陷产生原因

当所有的输入和数据都有效时，计算的最终结果也可以是无效的。所有变量都有值域范围，有时开发人员在执行计算时会忘记检查这些上限。

2）如何发现这类问题

一次又一次地执行计算或使用很大或很小的输入和数据进行计算，重点测试数据类型的初始值或边界值附近的值。

3）测试方法小结

（1）应用场合：应用程序执行能够导出待产生结果并进行内部存储的计算。

（2）测试方法：强制数据产生上溢或下溢。

（3）测试知识储备：全面掌握被测软件的需求，了解计算变量的上下限。

15. 数据共享或关联功能计算错误

1）缺陷产生原因

通常对孤立的功能进行测试时不会发生很多缺陷，而当把单独的功能和同一软件中的其他功能结合时，就可能出现很多软件缺陷。这种缺陷的产生往往是在两个或更多的功能使用了共享数据集，而每个功能允许使用的数据范围不同引起的。例如，一个功能可能会将某数据项设置为特定大小，然而另一个功能却允许该数据项的大小可以超过第一个功能的处理能力。开发人员根本没考虑到该数据项在其他功能处也可以修改，他们只是编码保证在该功能中数据的合法性，而当使用该数据时，没有再编码来检查可以使用的范围；而此时，另一个功能修改了共享数据，当再使用这些数据时就产生了缺陷。

2）如何发现这类问题

当应用程序在同一时间完成一个以上的功能或当一个以上的功能在同一时间处于运行状态时，就可以使用该方法进行测试。利用一个功能影响输入、输入数据或另一个功能的计算。在测试前要确定哪些功能是相互依赖或共享数据的。

(1) 能应用同样输入的每个功能。如果这些功能有相互重叠的输入域，就可能存在交互问题。

(2) 有类似的输出产生功能。如果某些功能结合起来产生单个输出，就说明这些部件之间存在关系，应该被一起测试。

(3) 一个功能被包含在另一个功能的计算中。例如，要测试鼠标选取对象的功能，不仅要测度鼠标选取屏幕上的文本的功能，还可以把包含超链接文本、粗体、斜体、符号及图形的元素放在一起，测试鼠标选取这些元素的功能。

3）测试方法小结

(1) 应用场合：一个以上的功能在同一时间处于运行状态。

(2) 测试方法：以点代面，重点测试某一功能，对可能与这个功能相连的其他功能附带测试。

(3) 测试知识储备：全面掌握被测软件的需求，在测试之前对被测功能之间的依赖关联有所掌握。另外，还需要对共享数据有所掌握。

16. 文件系统超载

1）缺陷产生原因

开发人员可能会忘记编写代码处理满状态的文件系统，忽略了诸如 CreateFile, WriteFile 等操作系统 API 的错误检查代码，没有这样的代码，当显示满状态的文件系统时，API 调用就会失败，软件就会在没有任何警告的情况下崩溃。

2）如何发现这类问题

创建满容量或近乎满容量的文件系统，然后强制执行各种通过输入或输出访问文件系统的操作；或者打开足够多的文件，打开文件时会强制备份创建的副本，从而占用双倍的存储空间，这种操作达到一定程度时，会达到该系统的容量，于是就能测试应用程序处理超载状态的文件系统的能力。(通常通过磁盘配额实现。)

3）测试方法小结

（1）应用场合：系统较大，运行时需要较大的空间。

（2）测试方法：强制磁盘系统满容量或容量小于等于被测软件运行时所需容量后，运行被测软件或利用测试工具模拟磁盘状况。

（3）测试知识储备：全面掌握被测软件的需求，了解被测软件处理超载状态的文件系统的能力。

17. 介质忙或不可用

1）缺陷产生原因

当多个应用程序同时访问硬盘（或其他存储器）时，操作系统为提供多请求服务会慢下来，并且必须对应用程序进行编程以处理这些延迟，当延迟变得很长时，没有对这些错误进行响应的应用程序就会出现错误。

2）如何发现这类问题

通过启动大量应用程序，强制它们都打开并保存文件使文件系统处理繁忙状态；或者同时下载大量文件也可以使后台拥挤；检查被测软件能否正确处理这种情况，应用程序应该给出错误信息或等待批示，提示用户正在处理。

3）测试方法小结

（1）应用场合：应用程序的运行需要消耗大量内存或运行时需要其他相关软件同时运行。

（2）测试方法：启动大量程序或利用测试工具模拟磁盘状况。

（3）测试知识储备：全面掌握被测软件的需求，了解被测软件运行时对系统的要求。

18. 介质损坏

1）缺陷产生原因

（1）损坏的介质可能会使操作系统传回错误代码，这些错误代码没有在应用程序中编程处理。

（2）操作系统不能检测出所有这样的错误，操作系统自己也有错误或者损坏的介质损坏了部分操作系统。

2）如何发现这类问题

使用损坏了的介质，例如刮伤、灰尘、磁干扰等。检查应用程序对错误的处理能力，应用程序可以对错误进行处理或者将问题告诉用户，并要确保用户数据文件不丢失、未损坏。

3）测试方法小结

（1）应用场合：应用程序对安全的要求较高，对灾难恢复的要求较高。

（2）测试方法：用实际损坏介质的方法测试应用程序。

（3）测试知识储备：全面掌握被测软件的需求，了解被测软件运行时对系统的要求。

19. 文件名不合法

1）缺陷产生原因

操作系统本身具有自己的文件命名规范，例如，DoS 的 8.3 格式。在 Windows 中，文件

名不能超过 255 个字符，并且文件名不可以含有/ \ ：＜ ＞ ？ ＊ |这 8 个字符，以及 AUX、COM1、COM2、COM3、COM4、CON、LPT1、LPT2、LPT3、LPT4、NUL 及 PRN 这些操作系统保留字。

开发人员在应用程序中使用不相同的规则管理文件名，当应用程序和操作系统使用的文件名命名规则不一致的时候，就会发生问题。

2）如何发现这类问题

（1）保存文件为操作系统不允许的文件名，例如，文件名中含有/ \ ：＜ ＞ ？ ＊ |这8 个字符，测试应用程序是否不允许输入包含这些字符的文件名。

（2）输入一些应用程序不允许使用的文件名，例如，使用过长的、含有特殊字符的、可能相互作用的字符作为文件名，检查应用程序能否识别该文件。

3）测试方法小结

（1）应用场合：几乎所有涉及需要输入文件名功能的应用程序。

（2）测试方法：输入操作系统不允许的文件名和应用程序不允许使用的文件名。

（3）测试知识储备：全面掌握被测软件的需求，了解操作系统和应用程序对文件名的要求。

20. 更改文件访问权限

1）缺陷产生原因

在操作系统中，可以设置不同用户对不同的文件具有不同的访问权限（如读写、只读等）。程序员必须在访问文件的函数中考虑文件的访问权限，例如，在每个文件写入之前检查文件的访问权限。如果没有进行检查，就会导致程序出错。另外，如果文件访问失败，程序员必须要有正确的错误代码，以保证程序可以正确捕获所产生的错误。

2）如何发现这类问题

（1）打开两个应用程序，关闭同一个文件。例如，把同一个应用程序的不同版本安装在同一机器上，在不同版本的应用程序中打开和关闭同一文件，或试着在某个应用程序中打开在另一个程序中已打开的文件，这可能导致文件访问权限的冲突。

（2）打开一个文件，在操作系统中修改文件的访问权限。有些操作系统允许权限高的用户控制一般用户已经打开的文件。

3）测试方法小结

（1）应用场合：需要对文件进行读写操作的应用程序。

（2）测试方法：修改文件访问权限或使用低权限的用户访问文件。

（3）测试知识储备：全面掌握被测软件的需求，了解读写文件所需的权限。

21. 文件内容受损

1）缺陷产生原因

开发人员编写代码来读取和写入文件，他们也编写代码来调用系统 API 得到文件指针，并打开和关闭文件。由于某些原因，这些系统 API 会失败或传回异常返回值。如果开发人员没有编写代码来验证传回的预期返回值，则应用程序会由于无法处理异常而失败。

2）如何发现这类问题

（1）手工损坏文件。从应用程序已创建的某个完整文件开始对其进行编辑，改变文件格式和内容。

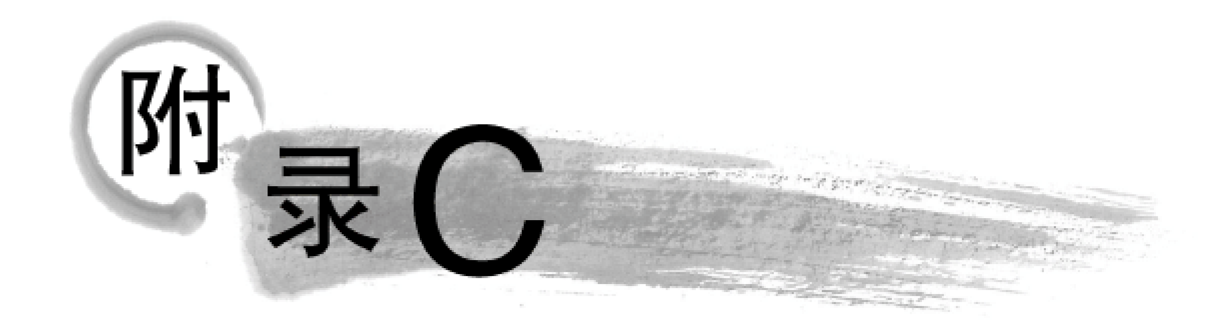
（2）使用测试工具。模拟 CRC（循环冗余校验）错误，或强制文件 API 返回无效的返回码。

3）测试方法小结

（1）应用场合：需要对文件格式和内容进行校验的应用程序。

（2）测试方法：手工损坏文件或利用测试工具模拟 CRC 错误。

（3）测试知识储备：全面掌握被测软件的需求，了解文件读写需要的权限。



软件评测师考试大纲

一、考试说明

1. 考试要求

(1) 熟悉计算机基础知识;

(2) 熟悉操作系统、数据库、中间件、程序设计语言基础知识;

(3) 熟悉计算机网络基础知识;

(4) 熟悉软件工程知识,理解软件开发方法及过程;

(5) 熟悉软件质量及软件质量管理基础知识;

(6) 熟悉软件测试标准;

(7) 掌握软件测试技术及方法;

(8) 掌握软件测试项目管理知识;

(9) 掌握C语言以及C++或Java语言程序设计技术;

(10) 了解信息化及信息安全基础知识;

(11) 熟悉知识产权相关法律、法规;

(12) 正确阅读并理解相关领域的英文资料。

2. 通过本考试的合格人员能在掌握软件工程与软件测试知识的基础上,运用软件测试管理办法、软件测试策略、软件测试技术,独立承担软件测试项目;具有工程师的实际工作能力和业务水平。

3. 本考试设置的科目包括:

(1) 软件工程与软件测试基础知识,考试时间为150分钟,笔试,选择题;

(2) 软件测试应用技术,考试时间为150分钟,笔试,问答题。

二、考试范围

考试科目1:软件工程与软件测试基础知识

1. 计算机系统基础知识

1.1 计算机系统构成及硬件基础知识

- 计算机系统的构成
- 处理机
- 基本输入输出设备
- 存储系统

1.2 操作系统基础知识
- 操作系统的中断控制、进程管理、线程管理
- 处理机管理、存储管理、设备管理、文件管理、作业管理
- 网络操作系统和嵌入式操作系统基础知识
- 操作系统的配置

1.3 数据库基础知识
- 数据库基本原理
- 数据库管理系统的功能和特征
- 数据库语言与编程

1.4 中间件基础知识

1.5 计算机网络基础知识
- 网络分类、体系结构与网络协议
- 常用网络设备
- Internet 基础知识及其应用
- 网络管理

1.6 程序设计语言知识
- 汇编、编译、解释系统的基础知识
- 程序设计语言的基本成分(数据、运算、控制和传输、过程(函数)调用)
- 面向对象程序设计
- C 语言以及 C++(或 Java)语言程序设计基础知识

2. 标准化基础知识
- 标准化的概念(标准化的意义、标准化的发展、标准化机构)
- 标准的层次(国际标准、国家标准、行业标准、企业标准)
- 标准的类别及生命周期

3. 信息安全知识
- 信息安全基本概念
- 计算机病毒及防范
- 网络入侵手段及防范
- 加密与解密机制

4. 信息化基础知识
- 信息化相关概念
- 与知识产权相关的法律、法规
- 信息网络系统、信息应用系统、信息资源系统基础知识

5. 软件工程知识

5.1 软件工程基础
- 软件工程概念
- 需求分析
- 软件系统设计
- 软件组件设计

- 软件编码
- 软件测试
- 软件维护

5.2 软件开发方法及过程

- 结构化开发方法
- 面向对象开发方法
- 瀑布模型
- 快速原型模型
- 螺旋模型

5.3 软件质量管理

- 软件质量及软件质量管理概念
- 软件质量管理体系
- 软件质量管理的目标、内容、方法和技术

5.4 软件过程管理

- 软件过程管理概念
- 软件过程改进
- 软件能力成熟度模型

5.5 软件配置管理

- 软件配置管理的意义
- 软件配置管理的过程、方法和技术

5.6 软件开发风险基础知识

- 风险管理
- 风险防范及应对

5.7 软件工程有关的标准

- 软件工程术语
- 计算机软件开发规范
- 计算机软件产品开发文件编制指南
- 计算机软件需求规范说明编制指南
- 计算机软件测试文件编制规范
- 计算机软件配置管理计划规范
- 计算机软件质量保证计划规范
- 数据流图、程序流程图、系统流程图、程序网络图和系统资源图的文件编制符号及约定

6. 软件评测师职业素质要求

- 软件评测师职业特点与岗位职责
- 软件评测师行为准则与职业道德要求
- 软件评测师的能力要求

7. 软件评测知识

7.1 软件测试基本概念

- 软件质量与软件测试
- 软件测试定义
- 软件测试目的
- 软件测试原则
- 软件测试对象

7.2 软件测试过程模型

- V 模型
- W 模型
- H 模型
- 测试模型的使用

7.3 软件测试类型

- 单元测试、集成测试、系统测试
- 确认测试、验收测试
- 开发方测试、用户测试、第三方测试
- 动态测试、静态测试
- 白盒测试、黑盒测试、灰盒测试

7.4 软件问题分类

- 软件错误
- 软件缺陷
- 软件故障
- 软件失效

7.5 测试标准

7.5.1GB/T 16260.1—2003 软件工程 产品质量 第 1 部分：质量模型

7.5.2GB/T 18905.1—2002 软件工程 产品评价 第 1 部分：概述

7.5.3GB/T 18905.5—2002 软件工程 产品评价 第 5 部分：评价者用的过程

8. 软件评测现状与发展

- 国内外现状
- 软件评测发展趋势

9. 专业英语

- 正确阅读并理解相关领域的英文资料

考试科目 2：软件测试应用技术

1. 软件生命周期测试策略

1.1 设计阶段的评审

- 需求评审
- 设计评审
- 测试计划与设计

1.2 开发与运行阶段的测试

- 单元测试
- 集成测试

- 系统(确认)测试
- 验收测试

2. 测试用例设计方法

2.1　白盒测试设计
- 白盒测试基本技术
- 白盒测试方法

2.2　黑盒测试用例设计
- 测试用例设计方法
- 测试用例的编写

2.3　面向对象测试用例设计

2.4　测试方法选择的策略
- 黑盒测试方法选择策略
- 白盒测试方法选择策略
- 面向对象软件的测试策略

3. 软件测试技术与应用

3.1　软件自动化测试
- 软件自动化测试基本概念
- 选择自动化测试工具
- 功能自动化测试
- 负载压力自动化测试

3.2　面向对象软件的测试
- 面向对象测试模型
- 面向对象分析的测试
- 面向对象设计的测试
- 面向对象编程的测试
- 面向对象的单元测试
- 面向对象的集成测试
- 面向对象的系统测试

3.3　负载压力测试
- 负载压力测试基本概念
- 负载压力测试解决方案
- 负载压力测试指标分析
- 负载压力测试实施

3.4　Web 应用测试
- Web 应用的测试策略
- Web 应用设计测试
- Web 应用开发测试
- Web 应用运行测试

3.5 网络测试

- 网络系统全生命周期测试策略
- 网络仿真技术
- 网络性能测试
- 网络应用测试

3.6 安全测试

- 测试内容
- 测试策略
- 测试方法

3.7 兼容性测试

- 硬件兼容性测试
- 软件兼容性测试
- 数据兼容性测试
- 新旧系统数据迁移测试
- 平台软件测试

3.8 易用性测试

- 功能易用性测试
- 用户界面测试

3.9 文档测试

- 文档测试的范围
- 用户文档的内容
- 用户文档测试的要点
- 用户手册的测试
- 在线帮助的测试

4. 测试项目管理

- 测试过程的特性与要求
- 软件测试与配置管理
- 测试的组织与人员
- 测试文档
- 软件测试风险分析
- 软件测试的成本管理

三、题型举例

(一) 选择题

1. 下面的哪一项测试步骤中需要进行局部数据结构测试?(　　)

   A. 单元测试　　B. 集成测试　　C. 确认测试　　D. 系统测试

2. 软件的 6 大质量特性包括(　　)。

   A. 功能性、可靠性、可用性、效率、可维护、可移植

   B. 功能性、可靠性、可用性、效率、稳定性、可移植

C. 功能性、可靠性、可扩展性、效率、稳定性、可移植

D. 功能性、可靠性、兼容性、效率、稳定性、可移植

（二）问答题

1. 白盒测试方法中的代码检查法需要重点考虑代码的执行效率，阅读以下两个循环，回答问题1和问题2。

循环1：

```
for (i = 0;i < n;i++)
{
if(condition)
DoSomething();
else
DoOtherthing();
}
```

循环2：

```
if(condition)
{
for (i = 0;i < n;i++)
DoSomething()
}
else
{
for (i = 0;i < n;i++)
DoOtherthing();
}
```

问题1：循环1的优点和缺点。

问题2：循环2的优点和缺点。

2. 请简述软件系统负载压力测试的主要目的。



附录D 软件评测师考试模拟试题

D.1 软件工程与软件测试基础知识

第 1 题至第 35 题为单选题，只有一个正确答案，每题 1 分；

第 36 题至第 50 题为多选题，有 2 或 3 个正确选项，至少有一个错误选项。多选，错选均不得分。少选，漏选按每个正确项得分 0.5 分。

一、单选题(35 小题，每题 1 分，共 35 分)

1. (　　)is measurable, verifiable work product such as specification, feasibility study report, detail document, or working prototype.

　　A. Milestone　　B. Deliverable　　C. Etc　　D. BAC

2. (　　)are individuals and organizations that are actively involved in the project, or whose interests may be affected as a result of project execution or project completion; they may also exert influence over the project and its results.

　　A. Controls　　B. Baselines

　　C. Project stakeholders　　D. Project managers

3. (　　) is the process of obtaining the stakeholdrs' formal acceptance of the completed project scope, verifying the scope includes reviewing deliverables and work results to ensure that all were completed satisfactorily.

　　A. Project acceptance　　B. Scope verification

　　C. Scope definition　　D. WBS Creation

4. In approximating costs, the estimator considers the possible causes of variation of the cost estimates, including (　　)。

　　A. budget　　B. plan　　C. risk　　D. contract

5. (　　)is a category assigned to products or services having the same functional use but different technical characteristics. It is not same as quality.

　　A. Problem　　B. Grade　　C. Risk　　D. Defect

6. 下列关于服务外包的说法，正确的是（　　）。

A. 服务外包是未来企业的发展趋势，其产生利润将超过产品贸易

B. 软件业服务外包依托互联网，可以加快产品的更新升级

C. 服务外包一般不需要中间的发包商，利于取得利润的最大化

D. 服务外包让包出企业专注核心技术，降低成本，提高效益

7. （　　）是一种面向数据流的开发方法，其基本思想是软件功能的分解和抽象。

A. 结构化开发方法　　B. Jackson 系统开发方法

C. Booch 方法　　D. UML

8. “<titlestyle="italic">science</title>”是一个 XML 元素的定义，其中元素标记的属性值是（　　）。

A. title　　B. style　　C. italic　　D. science

9. 选择测试开发工具时，应考虑功能、（　　）、稳健性、硬件要求和性能、服务和支持。

A. 易用性　　B. 易维护性

C. 可移植性　　D. 可扩充性

10. 在面向对象软件开发过程中，采用设计模式（　　）。

A. 允许在非面向对象程序设计语言中使用面向对象的概念

B. 以复用成功的设计和体系结构

C. 以减少设计过程创建的类的个数

D. 以保证程序的运行速度达到最优值

11. 某项目组准备开发一个大规模系统，且具备了相关领域及类似规模系统的开发经验。下列过程模型中，（　　）最适合开发此项目。

A. 原型模型　　B. 瀑布模型

C. V 模型　　D. 螺旋模型

12. 按照测试实施组织，可将测试划分为开发方测试、用户测试、第三方测试。下面关于开发方测试的描述正确的是（　　）。

① 开发方测试通常也叫“验证测试”或“Alpha 测试”

② 开发方测试又称“Beta 测试”

③ 开发方测试可以从软件产品编码结束之后开始，或在模块（子系统）测试完成后开始，也可以在确认测试过程中产品达到一定的稳定和可靠程度之后再开始

④ 开发方测试主要是把软件产品有计划地免费分发到目标市场，让用户大量使用，并评价、检查软件

A. ②③　　B. ①③　　C. ②④　　D. ①②③

13. 软件测试信息流的输入包括（　　）。

① 软件配置（包括软件开发文档、目标执行程序、数据结构）

② 开发工具（开发环境、数据库、中间件等）

③ 测试配置（包括测试计划、测试用例、测试驱动程序等）

④ 测试工具（为提高软件测试效率，使用测试工具为测试工作服务）

A. ①②③④　　B. ①②④　　C. ①③④　　D. ②③④

14. 下面关于软件测试模型的描述中,不正确的包括(　　)。

① V 模型的软件测试策略既包括低层测试又包括高层测试,高层测试是为了源代码的正确性,低层测试是为了使整个系统满足用户的需求

② V 模型存在一定的局限性,它仅把测试过程作为在需求分析、概要设计、详细设计及编码之后的一个阶段

③ W 模型可以说是 V 模型自然而然的发展。它强调:测试伴随着整个软件开发周期,而且测试的对象不仅是程序,需求、功能和设计同样要测试

④ H 模型中软件测试是一个独立的流程,贯穿产品整个生命周期,与其他流程并发地进行

⑤ H 模型中测试准备和测试实施紧密结合,有利于资源调配

A. ①⑤　　B. ②④　　C. ③④　　D. ②③

15. 系统功能测试过程中,验证需求可以正确实现的测试用例称为(　　)。

A. 业务流程测试用例　　B. 功能点测试用例

C. 通过测试用例　　D. 失败测试用例

16. 针对电子政务类应用系统的功能测试,为设计有效的测试用例,应(　　)。

A. 使业务需求的覆盖率达到 100%

B. 利用等价类法模拟核心业务流程的正确执行

C. 对一个业务流程的测试用例设计一条验证数据

D. 经常使用边界值法验证界面输入值

17. 白盒测试也称结构测试或逻辑驱动测试,典型的白盒测试方法包括静态测试和动态测试。其中,静态测试除了静态结构分析法、静态质量度量法外,还有(　　)。

A. 代码检查法　　B. 逻辑覆盖法

C. 基本路径测试法　　D. 结构覆盖法

18. 软件可移植性应从如下(　　)方面进行测试。

A. 适应性、易安装性、共存性、易替换性

B. 适应性、易安装性、可伸缩性、易替换性

C. 适应性、易安装性、兼容性、易替换性

D. 适应性、成熟性、兼容性、易替换性

19. 性能测试过程中需要对数据库服务器的资源使用进行监控,(　　)不属于应该监控的指标。

A. CPU 占用率　　B. 可用内存数

C. 点击率　　D. 缓存命中率

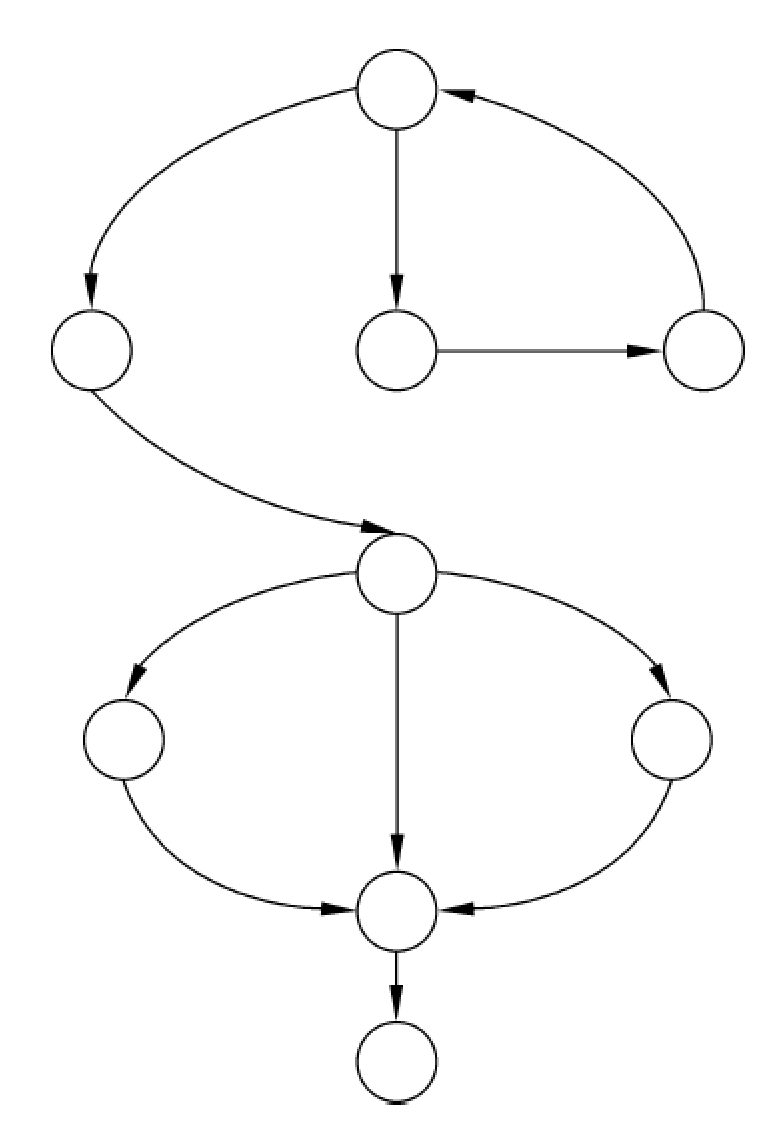
20. 功能测试执行过后一般可以确认系统的功能缺陷,缺陷的类型包括(　　)。

① 功能不满足隐性需求　　② 功能实现不正确

③ 功能不符合相关的法律法规　　④ 功能易用性不好

A. ①　　B. ①②③　　C. ②③④　　D. ②

21. 计算以下控制流程图的环路复杂性 $V(G)$，正确答案是（　　）。



A. $V(G)=2$　　B. $V(G)=4$　　C. $V(G)=9$　　D. $V(G)=11$

22. 软件评审作为质量控制的一个重要手段，已经被业界广泛使用。评审分为内部评审和外部评审。关于内部评审的叙述，正确的包括（　　）。

① 对软件的每个开发阶段都要进行内部评审

② 评审人员由软件开发组、质量管理和配置管理人员组成，也可邀请用户参与

③ 评审人数根据实际情况确定，例如根据软件的规模等级和安全性等级等指标而定

④ 内部评审由用户单位主持，由信息系统建设单位组织，应成立评审委员会

A. ①②④　　B. ①②③　　C. ②③④　　D. ①②③④

23. 测试记录包括（　　）。

① 测试计划或包含测试用例的测试规格说明

② 测试期间出现问题的评估与分析

③ 与测试用例相关的所有结果，包括在测试期间出现的所有失败

④ 测试中涉及的人员身份

A. ①②③　　B. ①③④　　C. ②③　　D. ①②③④

24. 对于监控测试周期时采用的度量方法，下列叙述中不当的是（　　）。

A. 基于故障和基于失效的度量：统计特定软件版本中的故障数

B. 基于测试用例的度量：统计各优先级的测试用例数量

C. 基于测试对象的度量：统计代码和安装平台等覆盖情况

D. 基于成本的度量：统计已经花费的测试成本，下一测试周期的成本与预期收益的关系

25. 下列（　　）是测试组独立的缺点。

A. 测试人员需要额外的培训

B. 测试人员需要花时间了解所要测试的产品的需要、架构、代码等

C. 开发人员可能会失去对产品质量的责任心

D. 设立独立测试组会花费更多成本

26. 事件报告中可能包括的错误有(　　)。

① 程序错误

② 规格说明中的错误

③ 用户手册中的错误

A. ①　　B. ①③　　C. ②③　　D. ①②③

27. (　　)是功能测试工具。

A. LoadRunner　　B. WinRunner　　C. QALoad　　D. WAS

28. 对需求说明书评测的内容包括(　　)。

① 系统定义的目标是否与用户的要求一致

② 被开发项目的数据流与数据结构是否足够、确定

③ 与所有其他系统交互的重要接口是否都已经描述

④ 主要功能是否已包含在规定的软件范围之内,是否都已充分说明

⑤ 确认软件的内部接口与外部接口是否已明确定义

A. ①③⑤　　B. ②③⑤　　C. ①②④⑤　　D. ①②③④

29. 在项目进行过程中,一个测试人员接到某个用户的电话,用户表明在系统中存在一个问题并要求更改,这个测试人员应该(　　)。

A. 马上通知程序员改正问题

B. 记录问题并提交项目经理

C. 不予理睬

D. 通过测试经理确认,确认后提交项目经理

30. 项目配置管理的主要任务中,不包括(　　)。

A. 版本管理　　B. 发行管理　　C. 检测配置　　D. 变更控制

31. 在开发的软件产品完成系统测试之后,作为最终产品应将其存入(　　),等待交付用户或现场安装。

A. 知识库　　B. 开发库　　C. 受控库　　D. 产品库

32. 为保证测试活动的可控性,必须在软件测试过程中进行软件测试配置管理,一般来说,软件测试配置管理中最基本的活动包括(　　)。

A. 配置项标识、配置项控制、配置状态报告、配置审计

B. 配置基线确立、配置项控制、配置报告、配置审计

C. 配置项标识、配置项变更、配置审计、配置跟踪

D. 配置项标识、配置项控制、配置状态报告、配置跟踪

33. (　　)可以作为软件测试结束的标志。

A. 使用了特定的测试用例

B. 错误强度曲线下降到预定的水平

C. 查出了预定数目的错误

D. 按照测试计划中所规定的时间进行了测试

34. (　　)是导致软件缺陷的最大原因。

A. 需求规格说明书　　B. 设计方案

C. 编写代码　　D. 测试计划

35. 下面①～④是关于软件评测师工作原则的描述，正确的判断是(　　)。

① 对于开发人员提交的程序必须进行完全的测试，以确保程序的质量

② 必须合理安排测试任务，做好周密的测试计划，平均分配软件各个模块的测试时间

③ 在测试之前需要与开发人员进行详细的交流，明确开发人员的程序设计思路，并以此为依据开展软件测试工作，最大程度地发现程序中与其设计思路不一致的错误

④ 要对自己发现的问题负责，确保每一个问题都能被开发人员理解和修改

A. ①②　　B. ②③　　C. ①③　　D. 无

二、多选题(15 小题，每题 1 分，共 15 分)

36. 对于工作规模或产品界定不明确的外包项目，一般不应该采用(　　)的形式。

A. 固定总价合同　　B. 成本补偿合同

C. 工时和材料合同　　D. 采购单

37. 软件项目的可行性研究要进行一次(　　)需求分析。

A. 详细的　　B. 全面的　　C. 简化的　　D. 压缩的

38. 下列属于动态测试方法的是(　　)。

A. 黑盒法　　B. 路径覆盖　　C. 白盒法　　D. 人工检测

39. 功能测试执行过后一般可以确认系统的功能缺陷，缺陷的类型包括(　　)。

A. 功能不满足隐性需求　　B. 功能实现不正确

C. 功能易用性不好　　D. 功能不符合相关的法律法规

40. 软件测试类型按开发阶段包括(　　)、系统测试和验收测试。

A. 单元测试　　B. 验证测试　　C. 集成测试　　D. 确认测试

41. 下列活动中，属于测试计划活动的是(　　)。

A. 设计测试用例　　B. 确定测试环境　　C. 定义测试级别　　D. 估算测试成本

42. 测试经理的任务通常包括(　　)。

A. 编写测试计划　　B. 选择合适的测试策略和方法

C. 建立和维护测试环境　　D. 选择和引入合适的测试工具

43. 配置管理系统通常由(　　)组成。

A. 基线库　　B. 动态库　　C. 主库　　D. 产品库

44. 缺陷管理中关于缺陷的严重程度的描述，包含(　　)。

A. 致命　　B. 严重　　C. 一般　　D. 可忽略

45. 一个团队从开始到终止，是一个不断成长和变化的过程，这个发展过程可以描述为4个时期，其中不包括(　　)。

A. 形成期　　B. 震荡期　　C. 完善期　　D. 优化期

46. 以下关于设计功能测试用例的叙述，(　　)是正确的。

A. 尽量用 80%的测试用例覆盖 20%的核心业务模块

B. 功能测试用例中不包括功能的依从性测试用例

C. 功能测试用例中包括业务流，也包括测试数据

D. 功能测试用例的设计应注意缺陷群集现象

47. 关于 Bug 管理流程,不正确的做法是(　　)。

A. 开发人员提交新的 Bug 入库,设置状态为“New”

B. 开发人员确认是 Bug,设置状态为“Fixed”

C. 测试人员确认问题解决了,设置状态为“Closed”

D. 测试人员确认不是 Bug,设置状态为“Reopen”

48. 测试成本控制的目标是使测试开发成本、测试实施成本和测试维护成本最小化,以下理解正确的是(　　)。

A. 测试准备成本不属于测试实施成本

B. 可以通过加强软件测试的配置管理来降低测试维护成本

C. 测试设计成本控制的目标是尽可能地减少测试总执行时间和所需的测试专用设备

D. 回归测试将测试案例全部重新执行一遍,可以将测试维护成本降至最低

49. 以下软件质量保证的目标中,(　　)是正确的。

A. 通过监控软件开发过程来保证产品质量

B. 保证开发出来的软件和软件开发过程符合相应标准与规程,不存在软件缺陷

C. 保证软件产品、软件过程中存在的问题得到处理,必要时将问题反映给高级管理者

D. 确保项目组制定的计划、标准和规程适合项目组需要,同时满足评审和审计需要

50. (　　)属于测试人员编写的文档。

A. 缺陷报告　　B. 测试环境配置文档

C. 缺陷修复报告　　D. 测试用例说明文档

D.2 软件测试应用技术

三、案例分析题(每小题 25 分,共 50 分)

试题一　阅读以下关于软件可靠性需求分析方面的叙述,回答问题 1、问题 2 和问题 3。

某企业信息部门的李工程师正在为其下属单位开发一个应用软件,在编写软件需求规格说明书时,涉及如何定量地描述软件可靠性的问题。

李工认为软件可靠性指的是在将要使用的指定环境下,软件能以用户可接受的方式正确运行任务所表现出来的能力。从定量角度看,似乎应当是该软件在约定的环境条件下和在给定的时间区间内,按照软件规格说明的要求,成功地运行程序所规定功能的概率。但是,他感到要具体地做定量描述有些困难。

为此,李工查阅到了本部门某个软件需求规格说明书中有关的一段内容:

(1) 在集成与系统测试期间,由非开发组人员参与测试,每 10k 行可执行代码可能检测到的错误(Bug)不能大于 6 个;

(2) 在提交使用的系统中,每 10k 行可执行代码可能保留的错误数不能大于 8 个;

(3) 在第一年工作期间,系统在 99.9%的工作日期间内,应能保持 100%的正常工作

状态。

在上述说明后，还有一条注解是：错误(Bug)可采用蒙特卡罗(MonteCarlo)随机植入技术进行测试

[问题1](12分)

李工程师首先想到了曾经学到过采用蒙特卡罗随机统计技术确定不规则形状封闭图形面积的方法，即采用一个大的矩形把待测的封闭图形完全包围在该大矩形的内部，由计算机大量生成在此矩形内均匀分布的"点"，然后，计数清点一下在大矩形内总的"点"的个数和在封闭图形内的"点"的个数，应当近似地有：

$$\text{封闭图形的面积}=\frac{\text{在封闭图形内的点的个数}\times\text{已知的大矩形的面积}}{\text{大矩形内总的点的个数}}$$

如果把这个思想应用于系统测试过程，先在某个程序中随机地人为植入10个错误(Bug)，然后，由一个测试组进行测试，结果一共发现有120个错误，其中有6个是人为植入的错误。

请估算一下这时该程序中将会遗留下多少个未被发现的隐藏错误。同时也请用100字以内的文字，简要地以提纲方式列举出采用这种错误随机植入方式来估算系统中遗留错误所固有的局限性。

[问题2](6分)

在进行上述分析后，李工程师感到有些困惑，于是与本企业维护系统的一位系统管理员进行了讨论，系统管理员告诉他可以借用硬件的MTTF(MeanTime To Failure，失效的平均等待时间)或者MTBF(MeanTime Between Failure，失效的平均间隔时间)作为软件可靠性的主要指标。

这时，李工程师查到了本企业中的一个典型例子：某软件在提交使用后，在第1周内有5次软件故障(查出了有关的Bug)，在第2～4周内共有两三次出错(也排除了错误根源)，在两个月以后该软件一直能正常使用运行(大家反映不错)，一直到6年半后的一天突然停工，即工作不正常。

请用100字以内文字分析该软件最后一次工作不正常的可能原因，并说明MTBF是在什么意义下反映了软件的可靠性。

[问题3](7分)

信息部门的吴总工程师向李工程师建议了另一类测试方案作为"错误随机植入"测试方法的补充。即由甲和乙两组测试人员同时相互独立地测试同一份程序，测试了两周后，甲组发现的错误总数为330个，乙组发现的错误总数为320个，其中两个组发现的相同错误数目为300个。请大体上估算一下在测试前此程序原有多少个错误？并也请以100字以内文字，简要说明使用这类估算方法的必要前提。

试题二 阅读下列说明，从项目测试管理和配置管理的角度，回答问题1至问题3。将解答填入答题纸的对应栏内。

【说明】

老高承接了一个信息系统开发项目的项目管理工作。在进行了需求分析和设计后，项目人员分头进行开发工作，期间客户提出的一些变更要求也由各部分人员分别解决。各部分人员在进行自测的时候均报告正常，因此老高决定直接在客户现场进行集成，但是发现问

题很多，针对系统各部分所表现出来的问题，开发人员又分别进行了修改，但是问题并未有明显减少，而且项目工作和产品版本越来越混乱。

【问题 1】(6 分)

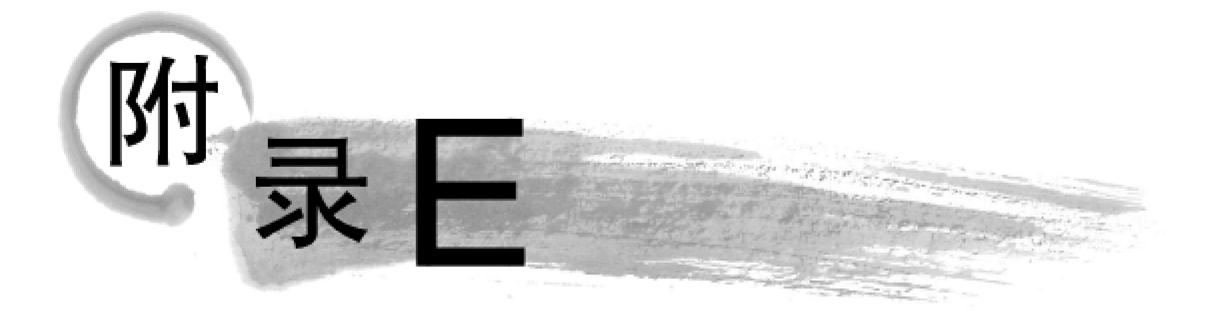
请用 200 字以内的文字，分析出现这种情况的原因。

【问题 2】(12 分)

请用 300 字以内的文字，说明配置管理的主要工作并做简要解释。

【问题 3】(7 分)

请用 300 字以内的文字，说明针对目前情况可采取哪些补救措施。



软件评测师考试模拟试题参考答案

E.1 软件工程与软件测试基础知识

一、单选题(35 小题,每题 1 分,共 35 分)

题号	选择项	答　　案
1	B.	B. Deliverable
2	C.	C. Project stakeholders
3	B.	B. Scope verification
4	C.	C. risk
5	B.	B. Grade
6	D.	D. 服务外包让包出企业专注核心技术,降低成本,提高效益
7	A.	A. 结构化开发方法
8	C.	C. italic
9	A.	A. 易用性
10	B.	B. 以复用成功的设计和体系结构
11	B.	B. 瀑布模型
12	B.	B. ①③
13	C.	C. ①③④
14	A.	A. ①⑤
15	B.	B. 功能点测试用例
16	D.	D. 经常使用边界值法验证界面输入值
17	A.	A. 代码检查法
18	A.	A. 适应性、易安装性、共存性、易替换性
19	C.	C. 点击率
20	B.	B. ①②③
21	B.	B. $V(G)=4$
22	B.	B. ①②③

续表

题号	选择项	答　　案
23	B.	B. ①③④
24	C.	C. 基于测试对象的度量：统计代码和安装平台等覆盖情况
25	C.	C. 开发人员可能会失去对产品质量的责任心
26	D.	D. ①②③
27	B.	B. WinRunner
28	D.	D. ①②③④
29	D.	D. 通过测试经理确认，确认后提交项目经理
30	C.	C. 检测配置
31	D.	D. 产品库
32	A.	A. 配置项标识、配置项控制、配置状态报告、配置审计
33	B.	B. 错误强度曲线下降到预定的水平
34	A.	A. 需求规格说明书
35	D.	D. 无

二、多选题（15 小题，每题 1 分，共 15 分）

多选，错选不得分，少选，漏选每项 0.5 分。

题号	选择项	答　　案
36	A B D	A. 固定总价合同　B. 成本补偿合同　D. 采购单
37	C D	C. 简化的　D. 压缩的
38	A C D	A. 黑盒法　C. 白盒法　D. 人工检测
39	A B C	A. 功能不满足隐性需求　B. 功能实现不正确　C. 功能易用性不好
40	A C D	A. 单元测试　C. 集成测试　D. 确认测试
41	B C D	B. 确定测试环境　C. 定义测试级别　D. 估算测试成本
42	A B D	A. 编写测试计划　B. 选择合适的测试策略和方法　D. 选择和引入合适的测试工具
43	B C D	B. 动态库　C. 主库　D. 产品库
44	A B C	A. 致命　B. 严重　C. 一般
45	C D	C. 完善期　D. 优化期
46	A C D	A. 尽量用 80%的测试用例覆盖 20%的核心业务模块　C. 功能测试用例中包括业务流，也包括测试数据　D. 功能测试用例的设计应注意缺陷群集现象
47	A B D	A. 开发人员提交新的 Bug 入库，设置状态为“New”　B. 开发人员确认是 Bug，设置状态为“Fixed”　D. 测试人员确认不是 Bug，设置状态为“Reopen”
48	A B	A. 测试准备成本不属于测试实施成本　B. 可以通过加强软件测试的配置管理来降低测试维护成本
49	A C D	A. 通过监控软件开发过程来保证产品质量　C. 保证软件产品、软件过程中存在的问题得到处理，必要时将问题反映给高级管理者　D. 确保项目组制定的计划、标准和规程适合项目组需要，同时满足评审和审计需要
50	A B D	A. 缺陷报告　B. 测试环境配置文档　D. 测试用例说明文档

E.2 软件测试应用技术

三、案例分析题(每小题25分,共50分)

试题一　答案

[问题1]　随机植入的局限性(12分)

程序总错误数约＝10×120/6＝200个,遗留未能发现的＝200－120－4＝76个。

(1) 所有错误不会平等地出现;

(2) 错误有连带相关性(一个错误的存在可能潜藏有另一个错误);

(3) 测试检测错误时,错误不是等同地可发现的;

(4) 修复错误常会引起新的错误。

评分标准:计算每个答案2分,要点每个2分。

[问题2]　MTBF指标的意义(6分)

突然停止正常工作的原因可能有:

(1) 用户突然启用一个以前从未用过的新功能(用户的可预测性);

(2) 某个软件维护人员犯了个错误,引入了一个新错误。

MTBF是反映“用户可预测性”与“软件中存在错误数”的一个复杂函数。

评分标准:原因每个2分,答对两个以上4分。MTBF解释2分。

[问题3]　两组相互独立预测方案(7分)

测试前程序原有错误可以认为是BNO＝320×330/300＝11×32＝352个。估算的前提:

(1) 前几周发现的错误在全部错误中有代表性;

(2) 两组发现的不同错误数所占比例相对很少。

评分标准:计算答案3分;估算前提每个2分,答对两个以上4分。

试题二　答案

【问题1】(6分)

(1) 缺乏项目整体管理(尤其是整体问题分析)。

(2) 缺乏整体变更控制规程。

(3) 缺乏项目干系人之间的沟通。

(4) 缺乏配置管理。

(5) 缺乏整体版本管理。

(6) 缺乏单元接口测试和集成测试。

评分标准:每个要点1分。

【问题2】(12分)

(1) 制定配置管理计划。确定方针,分配资源,明确职责,计划培训,确定干系人,制定配置识别准则,制定基线计划,制定配置库备份计划,制定变更控制规程,制定审批计划。

(2) 配置项识别。识别配置项,分配唯一标识,确定配置项特征,记录配置项进入时间,确定配置项拥有者职责,进行配置项登记管理。

(3) 建立配置管理系统。建立分级配置管理机制,存储和检索配置项,共享和转换配置项,进行归档、记录、保护和权限设置。

(4) 基线化。获得授权,建立或发布基线,形成文件,使基线可用。

(5) 建立配置库。建立动态库、受控库和静态库。

(6) 变更控制。包括变更的记录、分析、批准、实施、验证、沟通和存档。

(7) 配置状态统计。统计配置项的各种状态。

(8) 配置审计。包括功能配置审计和物理配置审计。

评分标准:每个要点 1.5 分。

【问题 3】(7 分)

1. 针对目前系统建立或调整基线;
2. 梳理变更脉络,确定统一的最终需求和设计;
3. 梳理配置项及其历史版本;
4. 对照最终需求和设计逐项分析现有配置项及历史版本的符合情况;
5. 根据分析结果由相关干系人确定整体变更计划并实施;
6. 加强单元接口测试与系统的集成测试或联调;
7. 加强整体版本管理。

评分标准:每个要点 1 分。

参 考 文 献

[1] 杜文洁,景秀丽.软件测试基础教程.北京:中国水利水电出版社,2008.

[2] 朱少民.软件测试方法和技术(第2版).北京:清华大学出版社,2010.

[3] 李炳森.软件质量管理.北京:清华大学出版社,2013.

[4] 李炳森.ITO接包操作实务.北京:中国商务出版社,2011.

[5] (美)项目管理协会.项目管理知识体系指南(PMBOK®)(第4版).王勇,张斌译.北京:电子工业出版社,2009.

[6] 覃征,徐文华,韩毅.软件项目管理(第2版).北京:清华大学出版社,2009.

[7] 左美云.信息系统项目管理.北京:清华大学出版社,2008.

[8] 彦哲研究院.信息化管理专家网. http://www.yima.org.cn.

[9] 李炳森.项目管理成功利器Project 2007全程解析:计划、管理和交流.北京:电子工业出版社,2008.

[10] 李炳森.Project 2007专案管理达人.中国台北:电脑人文化,2008.

[11] 李炳森.IT外包项目管理.北京:中国水利水电出版社,2014.

[12] 李炳森.软件外包质量管理.北京:中国水利水电出版社,2014.

[13] 李炳森.企业资产管理信息化.北京:中国水利水电出版社,2015.